{cvu}

Volume 18 Issue 2
April 2006
ISSN 1354-3164
www.accu.org

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

Contributors
Ian Bruntlett, Francis Glassborow, Lois Goldthwaite, Pete Goodliffe, Steve Hopley, Paul Johnson, Roger Orr, George Shagov, Örjan Westin

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

ACCU Secretary
Alan Bellingham
0117 942 7746
secretary@accu.org

ACCU Membership
David Hodge
0117 942 7746
membership@accu.org

Advertising
Thaddeus Froggley
ads@accu.org

ACCU Treasurer
Stewart Brodie
0117942 7746
treasurer@accu.org

Cover Art
Pete Goodliffe

Repro/Print
Parchment (Oxford) Ltd

Distribrution
Able Types (Oxford) Ltd

Design
Pete Goodliffe and
Alison Peck

accu

A brand new look!

Welcome to the redesigned C Vu! I've just seen the version about to be sent to the printers (well, an early draft) and I must say how impressed I am by it. Sure, I and a couple of others were consulted on the format and page style, but I shall now charge my glass to Pete and Alison for a job well done.

As always, the magazine is packed with the usual mixed bag of extremely high quality articles, editorial and masses of book reviews – and I mean it. Since I took over, you may have noticed that the number of reviews published has dropped somewhat due to the space being needed for articles. While it is encouraging to see the number of articles on the up, it's not as good that the reviews vanish, so for this and the next issue, I will be putting in all of the reviews dropped over the last 2 years (yes, it's been 2 years now!)

I think we're alone now (well, almost)

It's with sadness that I have to report the closure of CUJ. It is always sad when a magazine closes its door (for whatever reason) and with CUJ coming off the shelves, all that is left to the general public is Dr Dobbs, which is not exactly the easiest magazine to obtain (unless you live in London or close to Manchester Airport – I've certainly never seen it in Liverpool). It is a far cry from the halcyon days when magazines used to carry listings to help people to learn how to program and develop for their machines.

As you will see in the Secretary's report, there was talk of bringing the ACCU magazines to become a news stand publication. I don't have a problem with that, in fact, it would be a possible boon in some respects - there would be more pages, so I would have room to include some very large articles which have been submitted, but would require most of the magazine to bring to print as well as being able to having dedicated parts for C#, Agile development and if someone was to write it, parts for Java.

There would be logistical problems though – most notably would be that Overload and C Vu would need to merge to get the page count somewhere approaching viable and staffing would have to be arranged. While I would never want to impune on the Overload editorial team (or myself for that matter), but in my opinion, the new magazine would require a full time editor, much higher paper quality, a larger page count and probably the two largest hurdles: advertisers and more surprisingly, readers!

Of course, this has not been decided upon, so it's really academic . Personally, I'd love the CUJ writers to find a new home in Overload and C Vu – but I can see the logic behind moving the ACCU magazines to the news stands.

Ah well, that's enough for now – and on with the show!

[illegible]

PAUL JOHNSON,
EDITOR

The official magazine of the ACCU

The ACCU is an organisation of programmers who care about professionalism in programming. That is, we care about writing good code, and about writing it in a good way. We are dedicated to raising the standard of programming.

The ACCU exists for programmers at all levels of experience, from students and trainees to experienced developers. As well as publishing magazines, we run a respected annual developers' conference, and provide targeted mentored developer projects.

The articles in this magazine have all been written by ACCU members - by programmers, for programmers - and have been contributed free of charge.

To find out more about the ACCU's activities, or to join the organisation and subscribe to this magazine, go to www.accu.org.

Membership costs are very low as this is a non-profit organisation.

DIALOGUE

REGULARS

FEATURES

COPY DATES

C Vu 18.3: 1 May 2006
C Vu 18.4: 1 July 2006

IN OVERLOAD 72

Multithreading 101 by Tim Penhey, *To Grin Again* and *C++ Best Practice: Desigining Header Files* by Alan Griffiths, *A Fistful of Idiioms* by Steve Love and *Visiting Alice* by Phil Bass.

ADVERTISE WITH US

The ACCU magazines represent an effective, targeted advertising channel. 80% of our readers make purchasing decisions or recommend products for their organisations.

To advertise in the pages of C Vu or Overload, contact the advertising officer at ads@accu.org.

Our advertising rates are very reasonable, and we offer advertising discounts for corporate members.

COPYRIGHTS AND TRADE MARKS

Fuzzy Matching

Steve Hopley brings the 1890's US census into 2006.

Back in 1890, the USA was undergoing a census. Even at that time, it had a huge population and this would cause a massive logistical problem, not only because of the number of people but because of the English language. You see, the English language is very variable and because of that, there are many ways to spell the same (or very similar) surnames.

One solution was to match the sound of the word by using the "Soundex Coding" system. The big advantage is that similar sounding names have almost the same code sequence.

As with any form of sequence, it had its own rules.

1. Always retain the first letter of the word as the first character of the code.

From the 2nd letter on:

2. Ignore the vowels
3. Ignore *w*, *y*, *q* and *h*
4. Ignore punctuation
5. Code the other letters with the values 1 – 6 (*see table*)
6. Where adjacent letters have the same code, only the first is used
7. If the length of the code is greater than 4, only use the first 4 characters
8. If the length of the code is less than 4, pad it out with "0"

Letter	Code
b f p v	1
c g j k s x z	2
d t	3
l	4
m n	5
r	6

The upshot was that names like CUNNINGHAM would become C552 (C is retained, U, H , A and I are dropped, the first two Ns are given by 5 with the 3rd also being 5. G = 2 and M is 5. The final code is therefore C5525. As we only use the first 4 characters, this becomes C552.

HEYHOE gives the odd value of H000. H is kept and all the others are ignores. As this is only 1 character in length, the other three are padded with 0.

You can imagine that in 1890, the above method solved a lot of problems (and probably created many more). However, we can now perform this task quite simply and through a bit of code, we can generate Soundex Code from a standard input (see flowchart 1) and to start with, we need to define an array which holds the "allowed" characters in the order shown in the table. I've kept them in capitals purely for convenience.

```
string allowed[] = {"BFPV", "CGJKSXZ", "DT",
 "L",  "MN", "R"};
```

The code is read in `(via cin >> input)`. This needs encoding, so a variable of type string is used and the first character of input placed into it.

```
string encoded = input[0];
```

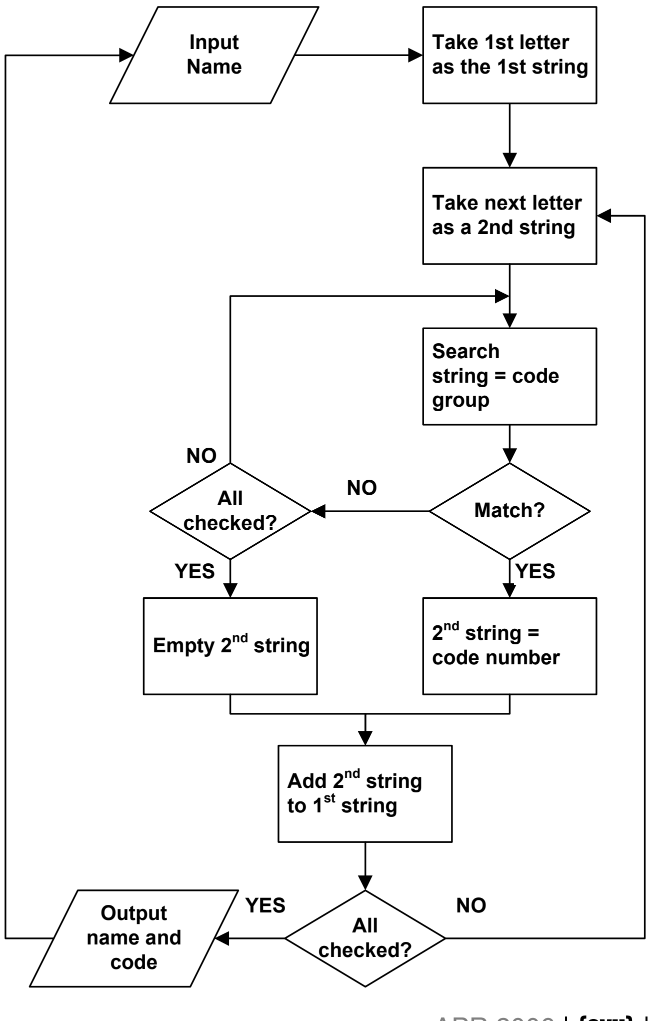
Next step is to read the rest of the string to the end of input into a temporary string. The temporary string contains only the letter to be tested

```
string temp;
for (int i = 1; i < input.length(); ++i)
  temp = input[i];
```

The conversion of code numbers will be required at various times in the program, so this is best farmed out to its own function (`int convertCodeNumbers(string t)`).

This function compares the values in the allowed array to the value passed in. If the letter is found, then the value of the loop where it occured is returned else `0`. Remember though, we have to add one to the return value if it's found.

```
int convertCodeNumbers(string t)
{
  for (int m = 0; m < 6; ++m)
  {
    if (allowed[m].find(t) != string::npos)
    return m + 1;
  }
  return 0;
}
```


Input Name
Take 1st letter as the 1st string
Take next letter as a 2nd string
Search string = code group
Match?
NO
YES
All checked?
NO
YES
Empty 2nd string
2nd string = code number
Add 2nd string to 1st string
All checked?
YES
NO
Output name and code

Flowchart 1

STEVE HOPLEY

Steve Hopley is a lecturer in Computer Systems at St Helens College of Further Education with special interests in cryptography and the deciphering of how a computer understands. He is a regular at Security and Anime conventions (though not at the same time!)
Steve can be contacted at shopley@sthelens.ac.uk

This will return a value of between 0 and 6 as an int which will need to be converted to a character if the value is > 0.

```
if (returned_value != 0)
  t += itoa(returned_value);
encoded += t;

...

string itoa(const int &value)
{
  ostringstream o;
  return (!(o << x)) ? "ERROR" : o.str();
}
```

Finally, you end up with source like listing 1.

This process is then repeated for the rest of the characters in the input string. A quick test should show that the name BRAIN gives B65 and CUNNINGHAM gives C55525. If you recall though, the output must be 4 characters in length, so the result for BRAIN would need padding with a "0" and CUNNINGHAM would require truncating.

CUNNINGHAM also exposes a problem in the code so far, it has given 555 which means that the two Ns have both been read (which isn't allowed).

Listing 1

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

using namespace std;

string allowed[6] = { "BFPV", "CGJKSXZ", "DT",
 "L", "MN", "R" };

int convertCodeNumbers(string t)
{
  for (int m = 0; m < 6; ++m)
    {
      if (allowed[m].find(t) != string::npos)
          return m + 1;
    }
  return 0;
}

string itoa(const int value)
{
  ostringstream o;
  return (!(o << value)) ? "ERROR" : o.str();
}

int main()
{
  string input, encoded, temp;
  char c;
  int rv;
  cout << "Please enter the name to encode : ";
  cin >> input;
  encoded = input[0];
  for (int n = 1; n < input.length(); ++n)
    {
      temp = input[n];
      rv = convertCodeNumbers(temp);
      if (rv > 0)
          encoded += itoa(rv);
    }
  cout << "Name          Code\n";
  cout << input <<  "      " << encoded << "\n";
}
```

The devil is in the detail

The repetition problem is solved by keeping a record of the last temporary string (**`string lastvalue`**, see flowchart 2). As with **`temp`**, the first character of the input string is placed in (to avoid it being repeated). This time in the iteration step, we compare **`lastvalue`** with temp and if they are the same, it's not added to encoded. If they are not the same, make **`lastvalue`** the current value of **`temp`** (listing 2).

Listing 2

```
#include <iostream>
#include <sstream>
#include <string>
#include <cstring>
using namespace std;
string allowed[6] = { "BFPV", "CGJKSXZ", "DT",
 "L", "MN", "R" };

int convertCodeNumbers(string t)
{
  for (int m = 0; m < 6; ++m)
    {
      if (allowed[m].find(t) != string::npos)
          return m + 1;
    }
  return 0;
}

string itoa(const int value)
{
  ostringstream o;
  return (!(o << value)) ? "ERROR" : o.str();
}

int main()
{
  string input, encoded, temp, copytemp;
  int rv;
  cout << "Please enter the name to encode : ";
  cin >> input;
  encoded = input[0];
  rv = convertCodeNumbers(encoded);
  copytemp = itoa(rv);
    for (int n = 1; n < input.length(); ++n)
    {
      temp = input[n];
      rv = convertCodeNumbers(temp);
      if (temp != copytemp)
      {
        copytemp = temp;
        if (rv > 0)
        encoded += itoa(rv);
      }
    }
  if (encoded.length() < 4)
    {
      for (size_t s = 4 - encoded.length();
       s < 4; ++s) encoded += "0";
    }
  if (encoded.length() > 4)
    {
      string t;
      t = encoded.substr(0, 4);
      encoded = t;
    }
  cout << "Name          Code\n";
  cout << input <<  "      " << encoded << "\n";
}
```

Once that has been done, we can deal with the case of the encoded string being less than or more than 4 characters by adding “0” or truncating respectively. Again, very simple

Matchmaker, matchmaker, make me a match

Okay. We now have a reliable system for producing the Soundex Codes and it would be quite trivial to replace the **cin >> input** with a list of words and examine their output. Again, I’ll just use an array, iterate through it and examine the results. The string below contains the data I’ve used to test my code with.

```
string tests[] = {"ABRAHAM", "ABRAHAMS",
   "ABRAMS", "ADAM", "ADAMS", "ADDAMS",
   "ADAMSON", "ALAN", "ALLAN", "ALLEN",
   "ANTHANY", "ANTHONY", "ANTONY", "ANTROBUS",
   "APPERLEY", "APPLEBY", "APPLEFORD" };
size_t arrayLen = sizeof(tests) /
sizeof(*tests);
for (int i = 0; i < t; ++i)
  {
     lastvalue = tests[i][0];
     convertCodeNumbers(tests[i]);
     ...
  }
cout << "NAME            CODE" << endl;
for (int i = 0; i < t; ++i)
  cout << test[i] + "        " + code[i] + "\n";
```

When compiled, the program produces:

NAME	CODE
ABRAHAM	A165
ABRAHAMS	A165
ABRAMS	A165
ADAM	A350
ADAMS	A352
ADDAMS	A352
ADAMSON	A352
ALAN	A450
ALLAN	A450
ALLEN	A450
ANTHANY	A535
ANTHONY	A535
ANTONY	A535
ANTROBUS	A536
APPERLEY	A141
APPLEBY	A141
APPLEFORD	A141

Now this is working, we can progress and perform a matching sequence whereby we find which codes of these names match the code from the input and then output these names, so that if APPLEBE was entered, the program would output answers.

```
NAME ? APPLEBE
```

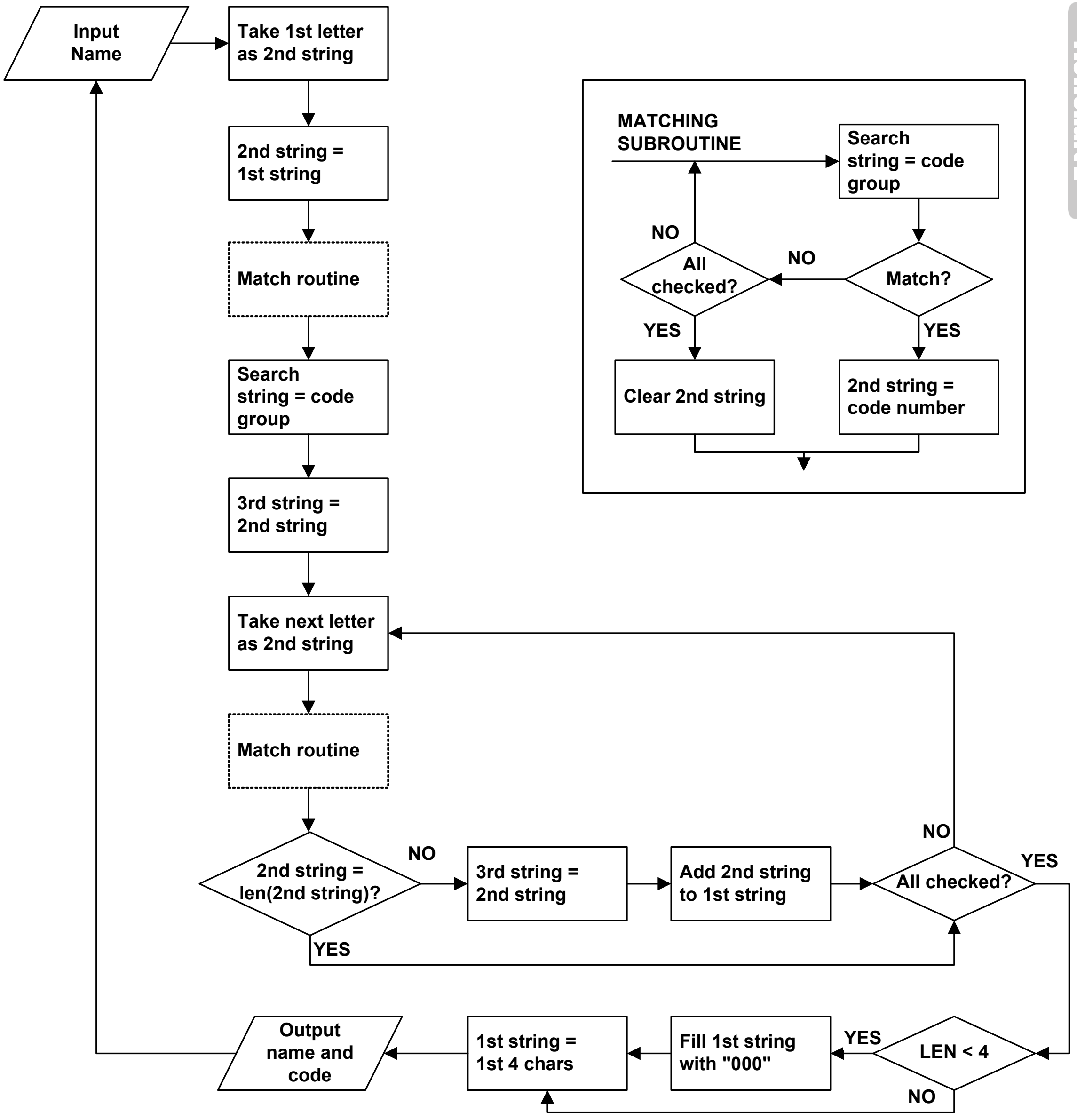
This is somewhat interesting as even though APPLEBE is not in the test list, the program has found the others listed.

Name	Code
APPLEBE	A141
APPLEBEE	A141
APPLEBY	A141
APPLEFORD	A141

Half a match is better than no match...

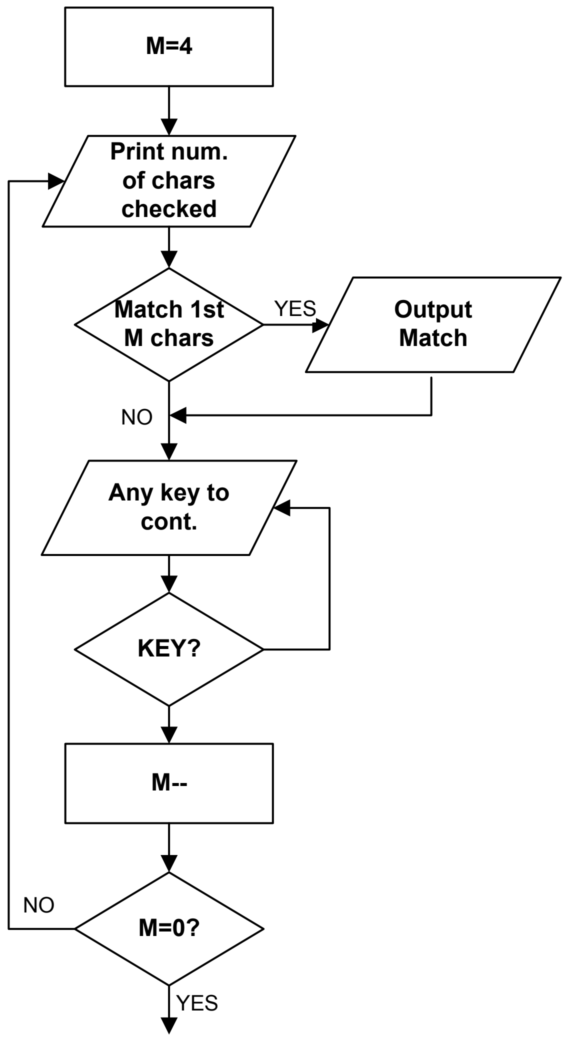
One thing which may have struck you about that last demonstration is that APPERLEY was rejected despite it sounding very similar. It would therefore be useful to be able to output these “near misses” as well. It is not hard to, just introduce another loop which compares a decreasing section of the input with decreasing lengths of the stored codes (flowchart 3).

If you were to enter APPLEBE, a whole range of answers are generated. Listing 3 gives such a possible solution. ■



Flowchart 2

Flowchart 3



Listing 3

```
#include <iostream>
#include <sstream>
#include <string>
#include <cstring>

using namespace std;

string allowed[6] = { "BFPV", "CGJKSXZ", "DT",
 "L", "MN", "R" };
string testdata[] = {"ABRAHAM", "ABRAHAMS",
 "ABRAMS", "ADAM", "ADAMS", "ADDAMS",
 "ADAMSON", "ALAN", "ALLAN", "ALLEN",
 "ANTHANY", "ANTHONY", "ANTONY", "ANTROBUS",
 APPERLEY", "APPLEBEE", "APPLEBY", "APPLEFORD"
};

int convertCodeNumbers(string t)
{
  for (int m = 0; m < 6; ++m)
    {
      if (allowed[m].find(t) != string::npos)
        return m + 1;
    }
  return 0;
}

string itoa(const int value)
{
  ostringstream o;
  return (!(o << value)) ? "ERROR" : o.str();
}
```

Listing 3 (continued)

```
string searchdata(string search)
{
  size_t arraylen = sizeof(testdata) /
   sizeof(*testdata);
  string input, encoded, temp, copytemp;
  int rv;

  for (int i = 0; i < arraylen; ++i)
    {
      input = search;
      encoded = input[0];
      rv = convertCodeNumbers(encoded);
      copytemp = itoa(rv);

      for (int n = 1; n < input.length(); ++n)
        {
          temp = input[n];
          rv = convertCodeNumbers(temp);

          if (temp != copytemp)
            {
              copytemp = temp;
              if (rv > 0)
                  encoded += itoa(rv);
            }
        }

     if (encoded.length() < 4)
        {
        for (size_t s = 4 - encoded.length();
         s < 4; ++s)
          encoded += "0";
        }
      if (encoded.length() > 4)
        {
          string t;
          t = encoded.substr(0, 4);
          encoded = t;
        }
    }
  return encoded;
}

int main()
{
  string test, data, search;
  cout << "Name to test : ";
  cin >> test;
  size_t arraylen = sizeof(testdata) /
   sizeof(*testdata);

  for (int m = 4; m > 0; --m)
    {
      cout << m << " characters match\n";
      for (int n = 0; n < arraylen; ++n)
        {
          if (testdata[n].substr(0, m) ==
           test.substr(0, m))
            cout << testdata[n] <<
             "              "
             << searchdata(testdata[n])
             << "\n";
        }
    }
}
```

Cygwin - Confessions of a Tool User

Ian Bruntlett on why using cygwin is not an ordeal.

I'm no spring chicken when it comes to using development or utility software. But when it came to downloading the Cygwin utilities, the rules changed. I've had mixed experiences with it. In the spirit of giving something back to the developers of the GNU tools and the people who port them to the Windows platform, here is an overview of my experiences.

In 1999 I was working on a database project (of the **`fopen`**, **`fseek`**, **`fread`** kind :) and we distributed it on DAT tapes to our customers. Previously we had used **`pctar`**, running under DOS 6.x. However, we shifted from working with DOS to working with Windows NT 4. I stumbled across the Cygwin tools and downloaded them and used them, no problem. We even quadrupled our throughput because instead of each PC writing one tape at a time, we had each PC writing four tapes at a time.

In Winter of 2005, I was doing voluntary work at a charity – Contact, `www.contactmorpeth.org.uk` – due to ill mental health. And I wanted to pass on knowledge about typical Linux tools – bash, gawk, perl, python, g++ as a payment in kind of the efforts the other volunteers were putting in on hardware. I would have to find and buy text books so I could teach myself how to use these tools properly. For technical help, I expected to rely on the ACCU.

I decided that I wanted to standardise on bash as my command line interpreter. So I wanted to install the Cygwin tools on the charity's computers, for use instead of `cmd.exe` or `command.com`. I tried over a period of a couple of weeks to download the files – but the Cygwin utilities had changed – instead of downloading lots of things by default, only the bare minimum would be. I couldn't get the hang of the Cygwin `setup.exe`. I looked at the Cygwin web site and FAQ, to no avail. Finally I sent a message to the accu-general mailing list and, with their help – and a bit of dumb luck, I worked out that the little round circles in each package line should be clicked on to cycle between different options like – ignore or install.

So my first suggestion as a grateful user of the Cygwin tools is to suggest that the FAQ have a section that shows people the little circle thing being used in practice. Believe me, this is a big help. To do otherwise, to use a British expression, is like putting a sign on your web site that says "Download our software if you think you are hard enough".

I am actually a big fan of `setup.exe` – in particular, the ability to download Cygwin via a broadband internet connection, burn it on to CDR and install it on many other computers without having to download it again is brilliant. Thank you for that, I really appreciate it.

My "second" suggestion is this: an update of the FAQ. Once someone has downloaded the tools, tell them about the GNU and O'Reilly books that have been published about the tools people have just downloaded. Just an entry in the FAQ would do. I've known about O'Reilly's books but I stumbled across the GNU books – and I hope to learn emacs, LISP and the gnu C++ compiler and debugger – amongst other things. APress is a good publisher but I'm not sure they've got any titles that are of interest to Cygwin users. I'm going to be learning from them this year and reviewing them for the ACCU's C Vu magazine.

One thing that caught me out when choosing Cygwin options for downloading is that there is no running counter of just how much information is going to be downloaded. My first download was a bit of a problem. To get downloads to work at all on Windows XP Home Edition, I had to be logged in as supervisor and I couldn't save it in a sub-folder below `My Documents`. To get it to work at all, I had to create a directory called `c:\DownloadCygwin`. I think the `My Documents` folder's full filename had characters that `setup.exe` just didn't like.

When deciding what to download, I make an informed decision based on: 1) how much space do I want to use on the charity's hard drives? 2) how much time do I have to download all this stuff? and 3) can I just download a selection of utilities?

When you are selecting individual packages for `setup.exe` to download, it tells you how big the individual packages are. However, I would like this facility to be expanded so that, say a running total of how many megabytes a particular category is going to have downloaded. And a final, overall figure, telling you how many megabytes you are going to download in this session.

On my most recent attempt to download the main GNU tools, I selected the tools I was interested in and then, to reduce the amount of time to download them, I went through the list of packages, de-selecting those that I thought were unlikely to be used by me in the foreseeable future and burnt them on to CDR. So far, so good. So I took the CDR home and put it into a Windows 2000 Professional PC donated by my sister's fiancé's father's business (I've kept Windows and MS Office on it because I teach people how to use that software as well as OpenOffice.org 2.x). I ran `setup.exe`, chose to install Cygwin from local directory.

And then I ran bash and tried to run different programming utilities. I got the message "bash: python: command not found" etc. So I thought I'd failed to download the right tools. I checked the CDR in Windows Explorer – the files were there except `setup.exe` overlooked them. I figured that `setup.exe` would just so a "install all the files I downloaded". I was wrong. The options that you select when you are downloading the files – you have to remember them so that, say when you come to install the software on a different PC, you must type in the options again. From memory.

So I tried again and got most, but not everything, working. In particular, running emacs gave the Windows dialogue box "emacs.exe - Unable To Locate DLL" The dynamic link library `cygungif-4.dll` could not be found in the specified path `C:\Cygwin\bin`; etc. This was particularly off-putting because I bought the GNU emacs books last year when I stumbled across the GNU shop site and I wanted to learn emacs LISP. I checked my CDR for `cygungif-4.dll` and it wasn't there. So the dependency management of `setup.exe` needs to be looked so that if you remove stuff from the download list, packages would be re-selected automatically to avoid dependency problems.

Cygwin is currently more difficult for me to install than, say some Linux distributions. This is an achievement in itself. Perhaps `setup.exe` could be changed to have profiles such as a) bare minimum command bash and tools, b) text mode programming (select languages required), c) text and GUI programming d) vim e) emacs f) text manipulation – groff etc. g) the full monty.

A further item on my wish list is to have a Cygwin icon on the Windows task bar at the bottom of the screen so I can quickly run a bash shell.

Finally, Contact has a Geordie computer – a Why Aye Mac. Known to the rest of the world as an i-mac. I want to run bash and g++ on the i-mac. Any suggestions are welcome. ■

www.cygwin.com offers GNU Unix and other tools ported to the Windows operating systems for Unix people to carry on using the tools they know and like, even on Windows.

IAN BRUNTLETT

Ian has been involved in broad spectrum of software systems and languages. He works as a volunteer, teaching people with mental health problems how to use and program computers. He can be reached at ianbruntlett@hotmail.com

The Curse of Good Code

Pete Goodliffe talks about documented, maintainable code.

Far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the Galaxy lies a small unregarded yellow sun. Orbiting this at a distance of roughly ninety-two million miles is an utterly insignificant little blue green planet whose ape-descended life forms are so amazingly primitive that they still think computer programs are a pretty neat idea.

This planet has - or rather had - a problem, which was this: most of the programmers on it wrote poor code pretty much all of the time, even when they were being paid to do a good job. Many solutions were suggested for this problem, but most of these were largely concerned with the education of programmers, which is odd because on the whole the programmers didn't want to be educated.

And so the problem remained; lots of the code produced was rubbish, and most of the users were miserable, even the ones who could write good computer programs.

Did you realise that as an ACCU member, and as a C Vu reader, you are an unusual beast? You're the kind of programmer who cares about the craft of programming, and who takes the initiative to improve their skills. Well done! Keep it up.

Anyone who's had the misfortune to recruit programmers will appreciate how rare a gift this is. Most CVs you'll read fail to demonstrate good programming practice and most 'C++ programmers' I've interviewed haven't the first clue about how to write correct code. They wouldn't recognise one end of a copy constructor from the other, even if it hit them over the head with a dangling pointer.

Yet these guys get hired, earn good money, and presumably churn out something resembling code. Eventually. It's not good code. It's not maintainable, extensible, bug-free, tested, and often not even correct (syntactically, semantically, aesthetically, emotionally, or theologically). You don't want to work in their wake; it's a messy place. Too many of us have had to suffer this kind of fate – mopping up after the inept.

Rant. Rant. Rant.

So what do we do about this? How can we save the (programming) world? That's a hard question. Good programmers don't grow on trees. How do you inform the bad apples that their skills suck? How do you convince them to care? And how do you get them to improve in a way that's agreeable and enjoyable? A philosophical conundrum indeed. There's one suggestion at the end of this article. But all *you* can do is strive to continually improve your own skills, to write excellent code, and to do your bit for reducing code entropy before Vogons blow up your codebase.

I'm always here to help, so here are a few more random disconnected thoughts about crafting high quality, professional, code. They're reminders for the indoctrinated, and sage lessons for the ill-informed.

Be conventional

I respect a man who knows how to spell a word more than one way.
– Mark Twain

This is one of the overlooked golden rules of programming. The following advice will lead to more comprehensible, easier to review code – and therefore to fewer faults and to better code quality. Sounds good, doesn't it?

write code without hidden surprises

There's also a hint of self-preservation here, and more than a little vanity, too. You've taken the time to write some code. You naturally want other people to use it, to extend it, to maintain it, and to enjoy it. You want them to bask in the magnificence of your coding genius. You don't want them to throw it all away, and replace it at their earliest convenience.

So the rule is simple: *be conventional*. Write code without hidden surprises, that works exactly as the reader would expect, that follows all the natural design and language idioms, and fits into the rest of the local codebase sympathetically.

Why does this matter?

- Favouring 'conventional' code prevents overly clever (and therefore overly complex) programs. This reduces potential bugs dramatically.
- You won't re-invent the wheel. Conventions are, by definition, tried and tested. They work – history has proved this. You'll conserve your precious brainpower, whilst writing code that's more likely to be correct.
- The reader (*a reviewer, maintenance programmer, or yourself in three years time*) can understand your code immediately. They can leverage their existing knowledge when working with the code, rather than have to figure it all out from first principles. They'll have more fun working with your code. If you annoy or surprise them, people will just walk away and find something easier to use. People don't accept (*or understand*) change.
- If someone can't immediately see what your code is doing, they'll modify it incorrectly and introduce a bug. Some time in the future this will haunt you – you just *know* that you'll get the blame.

What does this mean practically? Things like:

- Employ the coding and design idioms natural to your chosen language. [1]
- Use metaphors that others will understand, and pick variable/class names that make sense to the reader. [2]
- Use code presentation techniques that are commonplace. [3]
- Use standard tools to construct your code. [4] [5]
- Choose standard file names, with a convention for header files, source files, and build files. [2]
- Write GUI programs with standard keyboard short cuts.
- Consider how you package the code. Provide the README file that people expect and zip/tarball it in a subdirectory (so it unpacks predictably: there's nothing worse than unzipping a large file only for it to scatter files all over the current directory – what a mess!)

Sadly, this doesn't mean that you'll necessarily be writing the most exciting, cutting-edge code. We must compromise our desire to try out a new technique in order to write the highest quality software.

PETE GOODLIFFE

Pete Goodliffe is a programmer who never stays at the same place in the software food chain. He has a passion for curry and doesn't wear shoes. Pete can be contacted at pete@cthree.org

A case study: you are shipping a portable library written in C/C++. It must run on a variety of platforms. You want people to use it? Make a C++ idiomatic interface so that your users can learn the API easily, and so it can interoperate with other C++ libraries. Consider STL-like iterators as an interoperability mechanism. Provide makefiles to build the code (or on Unix use the standard autoconf / automake packages). Don't roll your own build system with a clever scripting language, or use a baroque new build tool just because it's cool. The user's effort to install the language or tool is probably not worth it. No one will use your library. Don't create artificial barriers and hurdles, you'll only hurt yourself [6].

So – should we abandon all innovation and keep doing the same old uninspired things over and over until the Vogons arrive? No, explore new techniques and idioms, but balance them against the needs of the programmers working with your code.

Too many programmers are shallow, uneducated, and terminally boring individuals

The next most important rule is probably: *be consistent*. Do all these conventional things consistently. But that's another story...

Write good documentation

Yes, it's hard to write, but it's harder not to.

– Carl Van Doren

This advice is born of frustration. My frustration. You might have just written the best code in the known world. It's elegant and efficient; a novel and beautiful solution to a tricky problem. But if I can't work out how to use it then it is *unusable* – practically useless.

When you write code, the user needs to be able to see clearly:

- what it can do,
- how to use it,
- where to find out more about it, and
- examples of usage.

You can do this through a number of means:

- "self documenting" code, [7]
- literate API programming tools, [7]
- sample code, and finally
- English documentation. [8]

Writing good documentation is not a God given gift. Every programmer must learn how to do it. It isn't easy, but since programming is all about communication, this is an essential skill.

A case study: I needed to use a particular facility in a popular C++ library. It was something that the library *must* have provided, but I simply couldn't find where it was or how to use it. Eventually, I gave up on the supplied docs and traced back through the library's labyrinthine header files to work out how to achieve my goal. It was painful experience which left me thinking a lot worse about that library.

Don't spoil your hard work with inadequate docs.

So what's worth documenting? Be careful about the assumptions you make here. What seems easy or obvious to you (someone well-versed in the code you're working with) may be incomprehensible gibberish to others. Writing 'conventional' code will reduce reliance on external documentation, but will not remove it.

Every time I explain a new principle to another programmer, I realise how much knowledge I've acquired over the years, and how much implicit knowledge I assume of other programmers. Be careful of making careless assumptions that render your docs (and code) worthless.

Great expectations

I have offended God and mankind because my work didn't reach the quality it should have.

– Leonardo da Vinci

This exercise is useful for a programmer, a manager, or a janitor. Just take a moment and check that you are delivering what is required of you. Do you meet expectations? This raises some interesting questions:

- What *do* people expect of you?
- How can they assess you? What's the scale?

Consider this for a minute, and then ask yourself: how well do I square up?

Do you deliver good code? Do you deliver comprehensive unit tests? [9] Do you deliver it all on time, repeatedly, and reliably? Do you provide adequate visibility of what you do? Do you inject many bugs into your code? Do you fix faults quickly? [10] Do you communicate well? Do you work with other team members well? [11] Is your design work up to scratch? [1]

And just who can tell? Is your manager / employer actually qualified to rate your programming skills? If not, how can you demonstrate that you are competent?

Think carefully about this stuff, it might help to secure your next raise.

Then ask the next question: do you *enjoy* what you're doing? Confucius said: *If you enjoy what you do you'll never work another day in your life.* If you don't: what's wrong? Is it something you can change at all?

Programming in the Real World

Everything we see is a perspective, not the truth.

– Marcus Aurelius

Too many programmers are shallow, uneducated, and terminally boring individuals. Sorry, you are! Here's a simple, but very important, piece of advice to help you become an exceptional software developer...

Develop outside interests that you can use as a frame of reference for technical knowledge. If all you ever study is programming then you will become a very two dimensional person, and will not be able to slot coding techniques into the context of the Real World.

There's a lot to learn outside your technical field that will teach you about what you do. Keep an open mind, and adopt a Real World perspective.

So long, and thanks for all the fish...

Not every end is the goal. The end of a melody is not its goal, and yet if a melody has not reached its end, it has not reached its goal.

– Friedrich Nietzsche

Go forth and write good code. Be professional. Digest these ideas, and apply them to the code you're writing right now. Never presume that you've reached the pinnacle of your programming career.

continued on page 10

Feedback

Thanks to all of the people who tell me that they've enjoyed my articles, have got something genuinely useful out of them, and have passed them on to colleagues for their 'education'. That's why I'm writing, and it makes a big difference to hear it. No one's paid to write in C Vu or Overload and *any* author appreciates feedback, from a simple encouragement to deep technical discussion. Even (constructive) criticism is better than silence. There's nothing worse than spending ages writing something for it to be *apparently* totally ignored!

So here's your challenge for the issue: If you've enjoyed reading anything in these pages, why not consider sending an email to the author to let them know. It really does make a big difference!

What Drives Development?

Phran Ryder looks at his own drivers.

My favourite conference is the BCS SPA conference (http://www.spaconference.org/spa2006/). I went to my first in 1994 and I have enjoyed them ever since as I learn so much. I learn so much because of the informal, interactive, group learning nature of the sessions. There arent many presenters who stand up and talk through PowerPoint presentations. Indeed, anyone who does wont be allowed to talk for long for he/she will be bombarded with questions and before long it will become a group debate. SPA stands for Software Practice Advancement, and of course BCS stands for British Computer Society.

Many years ago I found that if I wanted to be sure to get something out of a conference I would go to that conference with one or two questions I wanted to answer. At the SPA conference, the informal nature of the conference provides ample opportunities for my questions to be asked.

Last year my question was: What Drives Software Development? My reason for asking the question twofold: Firstly, I had recently read books on Test Driven Development, Domain Driven Development and Model Driven Development. Secondly, a senior, very experienced colleague had, at that time, said to a large audience: Everybody knows that a penny spent in design is worth a pound in development.

I was incredulous that someone still believed that such a thing was true. As a consequence I started an article, that I have not yet finished, which discussed why that statement was once true and why it no longer is true or at least shouldn’t be true. Whilst writing drafts of the article, a question kept coming back to me: What is it in software development that can at one time make it true that a penny in design is worth a pound in development?

By the end of the conference I had a list of some sixty answers, ranging from Ego driven development to Deadline Driven Development and Just F*****g do it Driven Development. I won’t waste column inches listing them all but I will ask you to take a few moments to think what drives your development?

Is it technology? Requirements? A trend? What is it the helps you decide what to do next? A process? A plan? A project manager? Feedback?

During the conference I coined one of my favourite phrases *Mortal Driven Development*. Partly this was in back lash to a perception of the type of people who embraced XP (eXtreme Programming). It seemed to me that the people who went into XP were highly talented, for want of a better expression, *Code Gods*. This does not seem right to me particularly as I am not a member of Valhalla, Olympus or wherever the Code Gods hang out. The thing is that to do XP properly there are certain things that you have to do to the extreme. The problem is that when push comes to shove mere mortals (like me) don’t - we just get the immediate job done. So in an environment of Pressure Driven Development we prefer Pragmatics Driven Development.

But by the end of the conference, and ever since, one thing alone is the biggest driving force to my development. If you have read, managed to finish, and remember any of my previous articles you may have guessed what this is...Value Driven Development. And in a larger font:

Value Driven Development

Every month, week, day, hour or minute I am consequently asking myself What will provide best value to the business. Try it! It helps me to decide what requirements to fulfil next, what test to write next, what proof of concept to do next, what design to do next&..Its is a great comfort blanket.

It also explains why a penny in design could be worth a pound in development. That philosophy originates in the early 1970s when studies were made into the cost of fixing programming defects. These showed that the cost of fixing defects rose exponentially over time. [Barry W. Boehm. *Software Engineering Economics*, Prentice Hall, 1981. ISBN 0138221227].

In the 1970s is might be hours or days before the code you wrote on punch cards was compiled. It might be longer before you were scheduled a slot to run your program. In short, the overheads and turn around times of development where very large. Thus, it paid to do heavy up-front design. The most valuable thing you could do was a good design. ■

PHRAN RYDER

Phran is Chairman of AgileNorth.org.uk (www.agilenorth.org.uk), a non-profit organisation for technical and business staff who wish to learn and share experience of becoming and being agile. He aspires to be a code God - contact him at phran@agilenorth.org.uk.

The Curse of Good Code (continued)

Continually seek to improve your coding skills and try to help others improve. Here’s one way that might work: roll this article up. Bash it over the head of the moron programmer behind you. Do it repeatedly, until they give in and start writing good code. Or until the spaceships arrive. ■

Notes and references

1. *Professionalism in programming #6: Good design*. In: C Vu 13.1
2. *Professionalism in programming #16: What's in a name?* In: C Vu 14.5
3. *Professionalism in programming #1: Layout of source code.* In: C Vu 12.2
4. *Professionalism in programming #8: The programmer's toolbox.* In: C Vu 13.3
5. *Professionalism in programming # 17: The code that Jack built.* In: C Vu 14.6
6. Exploiting your users’ laziness like this leads to another conclusion - don’t write a library that depends on too many other third party libraries! Too much installation effort will dissuade them from trying out your software. In this case, you are reinventing the wheel... Is this good advice? Work it out for yourself.
7. *Professionalism in programming #5: Documenting code.* In: C Vu 12.6
8. *Professionalism in programming #3: Being specific.* In: C Vu 12.4
9. *Professionalism in programming #10: Software testing.* In: C Vu 13.5
10. *Professionalism in programming #22: Finding fault.* In: C Vu 15.5
11. *Professionalism in programmer #34-36: Together we stand.* In: C Vu 17.5, 17.6, 18.1

Pete's new book, entitled Code Craft, is out soon. Keep your eyes peeled!

Shameless plug

I’m speaking at the 2006 ACCU conference in April. Come and hear me talk about *Effective version control*. You know you want to.

If you weren’t yet considering going to this event then I strongly encourage you to book a place. It’s excellent value, and a wonderful way to do what we’re attempting here – to learn how to improve your coding skills.

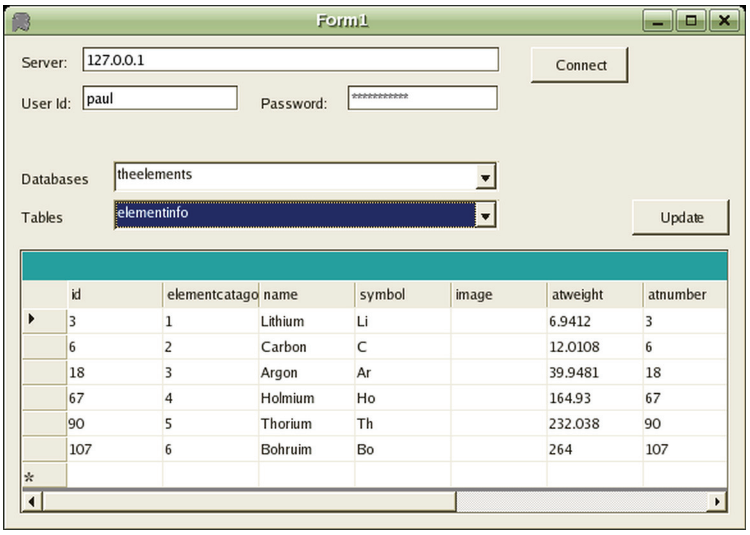
Design Time

Paul Johnson creates a GUI for his MySQL application.

In the beginning, life was simple. People had pieces of paper (or card) that contained words of wisdom, words of cheer and even sometimes, words which made sense. These words on the cards lived in TARDIS like structures which filled small rooms and would generally have misleading cards on the drawer fronts due to some dispute with the office girls over time off for hair combing (or something like that – I'm talking about the '70s when being in a union meant something!)

The system worked happily for many years. The TARDIS like structures creaked a bit, but something sinister was happening to the bits of card. Words began to vanish, fade or worse, form a blotchy mess due to the roof leaking. Could it be these pieces of paper/card were coming to the end of their regeneration cycle or something more sinister?

One day, many years after one man was nailed to the office memo board for suggesting that instead of mascara, the company should actually buy a new ribbon for the typewriter, a bright spark, was sitting in a dull room reading a copy of *Your Computer*. In it, amongst the adverts for a myriad of vapourware goodies and some very dodgy tape copying software (yeah right – "for backup purposes"), there was an article on modernising the office environment with this amazing new tool called a microcomputer. Sure, they were expensive, but compared to the amount of time it would take to find a card, ensure the details were correct and any other activity required to use it properly, they would soon start to *save* the company money. The bright spark spoke to an under manager, who spoke to a middle manger, who spoke to a higher manager, who spoke to the director who finally sought the approval of the second highest person in the company, the shop steward. The shop steward then grovelled at the heel of the real boss – the company secretary.



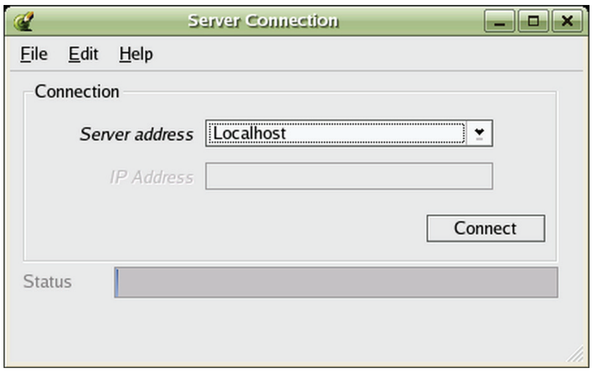
Server: 127.0.0.1
User Id: paul
Password: ***********
Databases: theelements
Tables: elementinfo

id	elementcatago	name	symbol	image	atweight	atnumber
3	1	Lithium	Li		6.9412	3
6	2	Carbon	C		12.0108	6
18	3	Argon	Ar		39.9481	18
67	4	Holmium	Ho		164.93	67
90	5	Thorium	Th		232.038	90
107	6	Bohruim	Bo		264	107

Figure 1

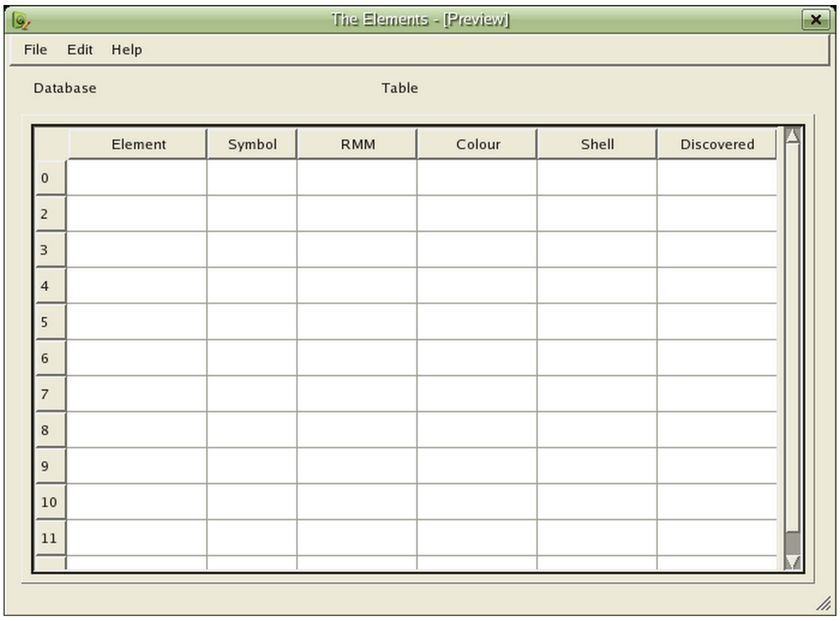
Eventually, the backend of the database system would be removed from the frontend and companies could have any old frontend as long as the frontend always communicated with the backend correctly. The frontend could be amazingly complex or stupidly simple – it didn't matter. What mattered was how the two communicated.

This is what we have now. MySQL provides the backend (server) and we can have what we want for the front end – which is what I'm looking at this time.



Server address: Localhost
IP Address:
Status:

Figure 2



	Element	Symbol	RMM	Colour	Shell	Discovered
0						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						

Figure 3

To cut a long story short, the secretary wanted an easy life and so the movement from bits of card to tape began. Millions of pieces of card would lose their jobs, but no-one would care. The Mecca which was technology had arrived.

There was only one problem though – there were lots of different computers out there and each had their own database software and usually, they would not talk to each other. This meant that if they moved from one machine to another, they'd still need to keep the old system running as the ability to translate the data from one format to another meant specialist software or worse, typing everything in again.

PAUL JOHNSON

Paul Johnson works at the University of Salford where he sometimes teaches, but mostly fixes computers and attempts to keep members of staff happy. He is currently looking for another job.

The Design

The first part of the design process is deciding what exactly I want the front end to do. Do I want something similar to that in figure 1, whereby everything is in one window or do I want something more like figures 2 and 3 which give two dialogs: one for the connection, the other for the data.

Both have their own advantages and disadvantages and both are simple to create using a development tool (such as MS VS.NET or Qt Designer [*used here*]). However, the key to what I want is simplicity.

Figure 1		Figures 2 and 3	
Advantages	**Disadvantages**	**Advantages**	**Disadvantages**
One window design Simple to use drop down for the table Data is editable	It looks unprofessional and is not obvious "Update" button isn't exactly clear as to it's purpose No undo facility No export facility	Clear and simple presentation Connection kept away from the main UI Undo facility Export facility Data is editable Connection can be dropped from main UI "File" menu	Two window system – increases code size Not as quick to navigate (tables are in a submenu in the "Edit" menu option.

Looking at the advantages/disadvantages, it is apparent that the second design is the one which I will go for.

Coding

Remember, this is a GUI application and in C#. Whilst there are the likes of gtk-sharp, Qt#, wx# and probably some others, it makes sense to stick with **System.Windows.Forms** as the widget set and as with all other widget sets, SWF has it's own events and event handlers. Happily, they are the same on all of the current C# platforms.

Window basics

If you're using VS.NET, you can ignore this. If you're using Mono and don't have access to VS.NET or a **System.Windows.Forms** designer, you need to pay attention – what follows is a whirlwind on setting up the windows without the designer.

The SWF window has a couple of components – a title bar and the window itself. Anything else is up to the designer.

The window is created like this:

```
using System.Windows.Forms;
using System.Drawing;
public class testwindow : forms
{
  public testwindow()
  {
    this.Size = new Size(400, 300);
    this.Text = "Some form title";
  }
  public static void Main()
  {
     testwindow t = new testwindow()
     Application.Run(t);
  }
}
```

Size is **(length, height)**. **Text** can be anything. This will give you a very simple window with nothing in it and a title bar. Impressive eh! Yeah, I thought you'd say that...

Let's go back to my original design (figure 2). The main difference this time is that I have some "bits" on the window, such as a writeable icon, a couple of labels, a status bad, a faded icon and a button (see Listing 1). These can be very easily added (with a bit of a change of code to help things become clearer – I've ignored the menus – they shouldn't be there!)

The widgets are declared to start off with. Prior though to the initialisation of the widgets, there is a piece of code : **protected override void Dispose(bool disposing)** this quite a common piece of code with SWF applications. It cleans up any resources being used nice and safely.

The main grunt work is now performed inside **InitializeComponent()**. To start with, the main window is created. There then follows the instantiation of the widgets:

```
this.Address = new Label();
// and so on for the other components
```

The positioning and setting up of the widgets occurs. This is where things become interesting if you're not using a designer!

```
this.Address.Size = new Size(32, 8);
this.Address.Name = "address";
this.Address.Text = "Server Address";
this.Address.Location = new Location(11, 8);
this.Address.TabIndex = 0;
```

Whilst the **Name** and **Text** entries are easy to understand and **Size** follows the same format for a widget as for a main window, **Location** refers to the *top left* corner of the widget in relation to the *top left* of the window; 11, 8 is 11 along and 8 down for the start of the widget.

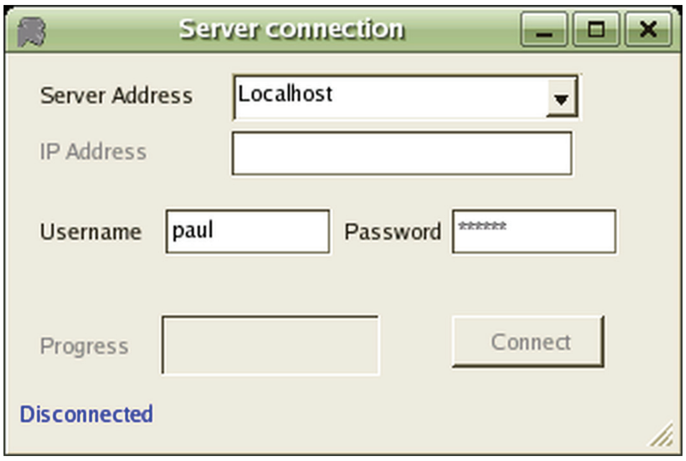
There are quite a number of changes from the simple version and it's not complete either! The new GUI looks like figure 4. It's not as attractive though.

You will note though that I have a drop-down box for the server address and that the progress label, progress bar and connection button are disabled. There is a good reason for that – unless the username and password have something in there or if an invalid IP address has been entered, I don't want anyone to be able to connect or even try to connect.

Listing 1

```
public class testwindow : forms
{
  private Label ServerAddress;
  private Label Address;
  private Label Status;
  private Button Connect;
  private ProgressBar Progress;
  private TextBox Username;
  private TextBox Password;
  private TextBox IPAddress;
  public testwindow()
  {
    InitializeComponent();
  }
  protected override void Dispose(
     bool disposing)
  {
    if (disposing)
    {
      if (components != null)
      {
         components.Dispose();
      }
    }
    base.Dispose(disposing);
  }
  private void InitializeComponent()
  {
    ...
  }
  public static void Main()
  {
     testwindow t = new testwindow()
     Application.Run(t);
  }
}
```

Figure 4



The problem now is that to re-enable the Connect button, ProgressBar or the IP Address, I have to add an event onto to the username/password writeable icons or the dropdown box respectively. Happily, it's simple to add.

For the dropdown box...

```
this.IPAddress.SelectedItemChanged +=
   new System.EventHandler(
   this.IPAddress_SelectIndexChanged);
```

For the writeable icons

```
this.Password.TextChanged +=
   new System.EventHandler(
   this.Password_TextChanged);
this.Username.TextChanged +=
   new System.EventHandler(
   this.Username_TextChanged);
```

The progressbar is controlled by the button, which itself needs an event – cunningly called **Click**.

```
this.Connect.Click +=
   new System.EventHandler(this.Connect_Click);
```

Events

Events are pretty simple to understand. You do something to a window – anything – and an event is generated (such as a window redraw, focus being gained or a whole plethora of other activities). By default, the only events that most windows have are resize and close with anything else the programmer should define. If a window was to act on all events, you'd soon end up in the situation whereby the machine grinds to a halt very quickly.

Event handling is a key aspect to any form of GUI application. If you consider an "old" command line program which used a simple menu, you would have a series of conditions on the menu which would (say) add a number to a value or save a file or something similar. The event in this case is the menu selection (it's arguable if it is an event really). With a GUI application, the simple "press 1" menu is now replaced with buttons, menus, text boxes and a pile of other bits and pieces. However, unless the program knows that clicking on the "Go" button means to do something, then it's eye candy – in effect, it's the same as having an unhandled menu item.

do something to a window – anything – and an event is generated

In the case of my little application, three types of events are used – **ButtonClick, SelectedItemChanged** and **TextChanged**.

Having added the previous code to the application, an event handler has to be written. These can be quite complex affairs.

The **TextChanged** events needs to check to see if the other writeable icon has some text in it before the button is enabled.

```
private void Password_TextChanged(
   object sender, System.EventArgs e)
{
  if (Username.Text.Length > 0
     && Password.Text.Length > 0)
     Connect.Enabled = true;
  else
     Connect.Enabled = false;
}
private void Username_TextChanged(
   object sender, System.EventArgs e)
{
  if (Username.Text.Length > 0
     && Password.Text.Length > 0)
     Connect.Enabled = true;
  else
     Connect.Enabled = false;
}
```

Similarly, the **SelectedItemChanged** needs to test to see which option has been selected as "Localhost" needs to disable to the **IPAddress** entry writeable icon.

```
private void IPAddress_SelectedIndexChanged(
   object sender, System.EventArgs e)
{
  Address.Enabled = IPAddress.SelectedIndex
     == 1 ? true : false;
  IP.Enabled = IPAddress.SelectedIndex
   == 1 ? true : false;
}
```

For completeness, I have a dummy **Button_Click** event defined.

```
private void Connect_Click(
   object sender, System.EventArgs e)
  {
  }
```

It may have occurred to you by now that having one large writeable icon for the IPAddress is not that good an idea. It's actually a very bad idea as the program would need to

1. Check that all the characters are either numbers or "."
2. Check that the numbers are all between 0 and 255
3. Check that there are 3 "." and 4 sets of numbers which can be of the form
   - 127.0.0.1
   - 127.000.000.001
   - 127.00.000.01
   - and any other numbers you fancy to put in

It is far simpler to just create 4, number only, writeable icons. Again, it's not hard to do – at the same time, I'll limit the maximum number of characters allowed to be 3.

```
private TextBox[] IP;
...
this.IP = new TextBox[4];
for (int i = 0; i < 4; ++i)
{
  // definitions
  this.IP[i].MaxLength = 3;
}
```

The problem comes with ensuring that only numbers can be entered. This can be done by extending the **TextBox** class:

```
class NumberBox : TextBox
{
  public NumberBox()
  {
    this.CausesValidation = true;
    this.Validating += new CancelEventHandler(
      TextBox_Validation);
  }
  private void TextBox_Validation(
     object sender, CancelEventArgs e)
  {
    try
    {
      int value = System.Int32.Parse(
         this.Text);
    }
    catch (System.Exception)
    {
      e.Cancel = true;
    }
  }
}
```

This gives me a final layout (figure 5) which I'm quite happy with. Sure, it's not as nice as the Qt mock up, but it's functional.

Listing 2 my final listing for the connection window. I'll extend it next time to include the main viewer window. The code at this stage is not optimal, nor is it bug free (the number range is still not as good as I'd like and the layout could do with some improvement), but it's making a start which is important. We've covered quite a bit here. Before hand, we had a nasty command line SQL interface. We still do, but at least now we're making headway into having a nice GUI.

```
// Paul's MySQL connector application - it is
// nearly complete it just needs the SQL
// connection code to be added which is very
// trivial - in fact, you do it and I'll show you
// how I did it next time!

// mcs connector.cs -r:
//   System.Windows.Forms -r:System.Drawing
// csc connector.cs -r:System.Windows.Forms -r:
//   System.Drawing

using System.Windows.Forms;
using System.ComponentModel;
using System.Collections;
using System.Drawing;

public class testwindow : Form
{
  private Label ServerAddress;
  private Label Address;
  private Label User;
  private Label Pass;
  private Button Connect;
  private TextBox Username;
  private TextBox Password;
  private ComboBox IPAddress;
  private NumberBox[] IP;

  private StatusBar StateBar;
  private ProgressBar Progress;
  private Label Connection;
  private System.ComponentModel.
     Container components = null;
```

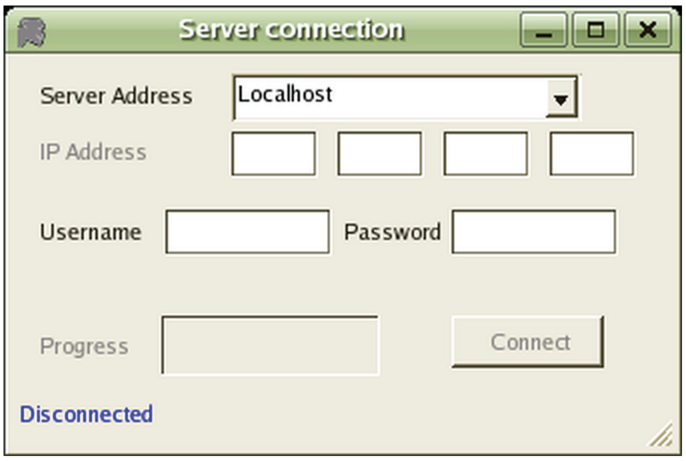


Figure 5

```
private testwindow()
{
  InitializeComponent();
}

protected override void Dispose(
   bool disposing)
{
  if (disposing)
  {
    if (components != null)
    {
      components.Dispose();
    }
  }
  base.Dispose(disposing);
}

private void InitializeComponent()
{
  this.SuspendLayout();

  this.Size = new Size(300, 150);
  this.Text = "Server connection";

  this.ServerAddress = new Label();
  this.ServerAddress.Size = new Size(72, 12);
  this.ServerAddress.Name = "address";
  this.ServerAddress.Text = "Server Address";
  this.ServerAddress.Location =
     new Point(11, 10);
  this.ServerAddress.TabIndex = 0;

  this.IPAddress = new ComboBox();
  this.IPAddress.DropDownStyle =
     ComboBoxStyle.DropDownList;
  this.IPAddress.BackColor = Color.White;
  this.IPAddress.ForeColor = Color.Black;
  this.IPAddress.Size = new Size(160, 8);
  this.IPAddress.Name = "ipaddress";
  this.IPAddress.Location = new Point(100, 8);
  this.IPAddress.Items.Add("Localhost");
  this.IPAddress.Items.Add("Other");
  this.IPAddress.SelectedIndex = 0;
  this.IPAddress.SelectedIndexChanged +=
     new System.EventHandler(
     this.IPAddress_SelectedIndexChanged);
  this.IPAddress.TabIndex = 1;

  this.Address = new Label();
  this.Address.Size = new Size (72, 12);
```

```
    this.Address.Name = "numaddr";
    this.Address.Text = "IP Address";
    this.Address.Enabled = false;
    this.Address.Location = new Point(11, 32);
    this.Address.TabIndex = 2;
    this.IP = new NumberBox[4];
    for (int i = 0; i < 4; ++i)
    {
      this.IP[i] = new NumberBox();
      this.IP[i].Size = new Size(44, 8);
      this.IP[i].Enabled = false;
      this.IP[i].Location = new Point(100 +
         (48 * i), 30);
      this.IP[i].TabIndex = 3 + i;
      this.IP[i].MaxLength = 3;
    }

    this.User = new Label();
    this.User.Size = new Size(54, 12);
    this.User.Name = "user";
    this.User.Text = "Username";
    this.User.Location = new Point(11, 62);
    this.User.TabIndex = 7
;
    this.Username = new TextBox();
    this.Username.Size = new Size(80, 8);
    this.Username.Name = "username";
    this.Username.Location = new Point(70, 60);
    this.Username.TextChanged +=
       new System.EventHandler(
       this.Username_TextChanged);
    this.Username.TabIndex = 8;

    this.Pass = new Label();
    this.Pass.Size = new Size(48, 12);
    this.Pass.Name = "pass";
    this.Pass.Text = "Password";
    this.Pass.Location = new Point(150, 62);
    this.Pass.TabIndex = 9;

    this.Password = new TextBox();
    this.Password.Size = new Size(80, 12);
    this.Password.Name = "password";
    this.Password.PasswordChar = (char)'*';
    this.Password.Location =
       new Point (200, 60);
    this.Password.TextChanged +=
       new System.EventHandler(
       this.Password_TextChanged);
    this.Password.TabIndex = 10;

    this.Connect = new Button();
    this.Connect.Size = new Size(70, 20);
    this.Connect.Name = "connect";
    this.Connect.Enabled = false;
    this.Connect.Text = "Connect";
    this.Connect.Location =
       new Point (200, 100);
    this.Connect.Click +=
       new System.EventHandler(
       this.Connect_Click);
    this.Connect.TabIndex = 11;

    this.StateBar = new StatusBar();
    this.StateBar.Location = new Point(0, 130);
    this.StateBar.Height = 20;
    this.StateBar.Name = "status";
    this.StateBar.ForeColor = Color.Blue;
    this.StateBar.Text = "Disconnected";
    this.StateBar.TabIndex = 12;

    this.Connection = new Label();
    this.Connection.Size = new Size(50, 12);
    this.Connection.Text = "Progress";
    this.Connection.Enabled = false;
    this.Connection.Name = "connprog";
    this.Connection.Location =
       new Point(11, 106);

    this.Progress = new ProgressBar();
    this.Progress.Location = new Point(68,100);
    this.Progress.Minimum = 0;
    this.Progress.Step = 1;
    this.Progress.Maximum = 10;
    this.Progress.Name = "progbar";
    this.Progress.Enabled = false;

    this.AutoScaleBaseSize =
       new System.Drawing.Size(5, 13);
    this.Controls.AddRange(new Control[] {
       this.ServerAddress, this.IPAddress,
       this.IP[0], this.IP[1], this.IP[2],
       this.IP[3], this.User, this.Username,
       this.Pass, this.Password, this.Connect,
       this.Connection, this.StateBar,
       this.Progress, this.Address});
    this.ResumeLayout();
  }

  public static void Main()
  {
    testwindow t = new testwindow();
    Application.Run(t);
  }

  private void Password_TextChanged(
     object sender, System.EventArgs e)
  {
    if (Username.Text.Length > 0
       && Password.Text.Length > 0)
      Connect.Enabled = true;
    else
      Connect.Enabled = false;
  }
  private void Username_TextChanged(
     object sender, System.EventArgs e)
  {
    if (Username.Text.Length > 0 &&
       Password.Text.Length > 0)
      Connect.Enabled = true;
    else
      Connect.Enabled = false;
  }
  private void IPAddress_SelectedIndexChanged(
     object sender, System.EventArgs e)
  {
    Address.Enabled = IPAddress.SelectedIndex
       == 1 ? true : false;
    for (int i = 0; i < 4; ++i)
    {
      IP[i].Enabled = IPAddress.SelectedIndex
         == 1 ? true : false;
    }
  }
  private void Connect_Click(
     object sender, System.EventArgs e)
  {
  }
}

class NumberBox : TextBox
{
  public NumberBox()
  {
    this.CausesValidation = true;
    this.Validating += new CancelEventHandler(
       TextBox_Validation);
  }

  private void TextBox_Validation(
     object sender, CancelEventArgs ce)
  {
    try
    {
      int value = System.Int32.Parse(this.Text);
    }
    catch(System.Exception)
    {
      ce.Cancel = true;
    }
  }
}
```

Next time

Next time, I'll continue with the WinForms code for the main GUI and also demonstrate how to show the data from the MySQL server. What I've covered here is just a beginning...

Thanks

I must thank two people for helping me with this article and one organisation – the first is an old friend of mine, Pat de Ridder. He had a good look at the code and at what I was trying to do and made quite a number of useful suggestions. He also provided quite a nice way of ensuring that only numbers are allowed into the textboxes – unfortunately, I couldn't get it to work (must be a difference in operation between Mono and VS.NET)

The other is my son, Richard who has been testing the code for a month or so now and has also been entering a lot of data for me (not bad considering he's only 7!).

I'm also thankful for the support from Cansfield High Specialist Language College, Ashton-in-Makerfield for providing the inspiration for a code redesign one lunchtime. ■

With so many books, why are there so many gaps?

While writing part 3 of this series, I discovered something – and it's something that I'm finding happening more and more over time.

I often say to people when I'm developing software that "*I love the engine and hate the eye candy*" - in other words, give me the raw code and I'm fine. Tell me to interact with a GUI (using any widget set) and I'm not so much lost, but I do have problems. It's not that I don't understand how GUIs work using the usual event / message system, it's just that they are so badly documented in books.

Take System.Windows.Forms. There are a large number of books available, of differing quality, but all of which make the same mistake. They assume that you're using an IDE or some form of window designer code. The programmer really doesn't need to do much, set a couple of events and let the form designer take care of the rest. They have to program the event handlers, but even they can be prototyped in the form code, so it's a case of insert code here.

I have yet to come across a book which starts using first principles for SWF (or many of the other widget sets available) and builds up from there. I can't blame the authors, as for the majority of them they are targeting people using VisualStudio.NET – which the largest section of the audience is. The target audience may not even be programmers, but want to hack something together to do a particular job.

However, for the likes of me (using Mono) or anyone who is making use of the free version of the C# compiler from Microsoft, these books aren't much use and so we have to rely on the internet or MSDN or mailing lists (such as the ACCU general or programmer list). Good as they may be, if you're on a train or have a spare 20 minutes to carry on with a program, they're no substitute for a good book.

It gets worse though if you try to search on the net for what you're looking for. While writing the article, I needed to find the name for one particular generic, but didn't know what it was called. I tried for about 20 minutes and gave up – the worst of it was that I had found the answer on about the third hit, but because of the description being purely textual, it didn't ring any bells.

Acorn had the right idea with their Programmers' Reference Manuals, specifically the toolbox manual. This book gave every widget, their events, the messages, parameters in and out, the lot. It wasn't a programming book by any stretch of the imagination, but by having it close to hand, it was usually very simple to work out how to code the application and best of all, it had pictures.

Even the likes of Brown's *Windows Forms Programming with C#*, one of the finest C# books around in my opinion, seems to miss this. Don't anyone mention MSDN. Sure, it has code examples, but it's a hideous mess!

Books now seem to be aimed at those who just want to use the IDE to create an image viewer or those who are into full sized mega applications with very little in between. Even seemingly simple jobs, such as forms talking to other forms, multiple windows, child – parent relationships seem to be overlooked.

If anyone has a suggestion for a book which will fill these gaps, please let me know. I don't make any claims to having read every book.

C++ Forever

George Shagov looks at transferring data across a network.

`Come back!' the Caterpillar called after her.
`I've something important to say!'
This sounded promising, certainly: Alice turned and came back again.
`Keep your temper,' said the Caterpillar.
`Is that all?' said Alice,swallowing down her anger as well as she could.
`No,' said the Caterpillar.
Lewis Carroll, Alice's Adventures in Wonderland

Phase I: Life story

Being on the road all the time, it is quite important to stop and to think:

1. Where are we coming from?
2. Where are we now?
3. Where are we going to?

Preamble

I have a friend who works as a C++ server developer for a large respectable company. He told me this story, which is quite interesting from a professional point of view.

The application being developed is a C++ server on an UNIX-like platform. Inter-Process-Communication is done through sockets and it uses some third-party product for the communication layer. Let me title everything related to this library within the **`net-`** prefix, like **`net-library`**, **`net-API`**, **`net-object`**, etc. The **`net-API`** of this library constructs a **`net-object`** and fills it by means of the fields, where each particular field is represented by its name. When the **`net-object`** is constructed and filled, it is going to be populated on a network. For instance if we have a **`cpp-class`**:

```
class CAuthor
{
public:
  int m_nID;
  std::string m_strFirstName;
  std::string m_strLastName;
  std::string m_strPhone;
  /*
   * and so on, <skipped>
   */
;
```

then sending this class through the network might seem like this:

```
void CAuthor::Send()
{
  Net_Object netObj =
   net_API_ConstructNetObject();
  net_API_AddIntField(netObj, "id", m_nID);
  net_API_AddStringField(netObj, "firstname",
   m_strFirstName);
  net_API_AddStringField(netObj, "lastname",
   m_strLastName);
  net_API_AddStringField(netObj, "phone",
   m_strPhone);
  /*
   * and so on, <skipped>
   */
  net_API_SendNetObjcet(netObj);
}
```

and of course there should be a procedure that ‘receives’ objects from the network.

In order to avoid the tediousness of putting these fields into the net-object, additional macros have been written. Therefore the declaration of the cpp-class looks like this:

```
class CAuthor
{
public:
  BEGIN_ENTRY_MAP(CAuthor)
    ENTRY_DECL("id", m_nID);
    ENTRY_DECL("firstname", m_strFirstName);
    ENTRY_DECL("lastname", m_strLastName);
    ENTRY_DECL("phone", m_strPhone);
    /*
     * and so on, <skipped>
     */
  END_ENTRY_MAP;

public:
  int m_nID;
  std::string m_strFirstName;
  std::string m_strLastName;
  std::string m_strPhone;
  /*
   * and so on, <skipped>
   */
};
```

The macro **`BEGIN_ENTRY_MAP`** instantiates some static function within the static map of the fields and **`ENTRY_DECL`** adds one row to that static map. The first parameter of the macro is the name of the net-field, required by **`net-API`**, the second parameter is the field value itself.

Each class has a **`Copy`** member, in order to apply the fields which come from network to the existing cpp-object, something like this:

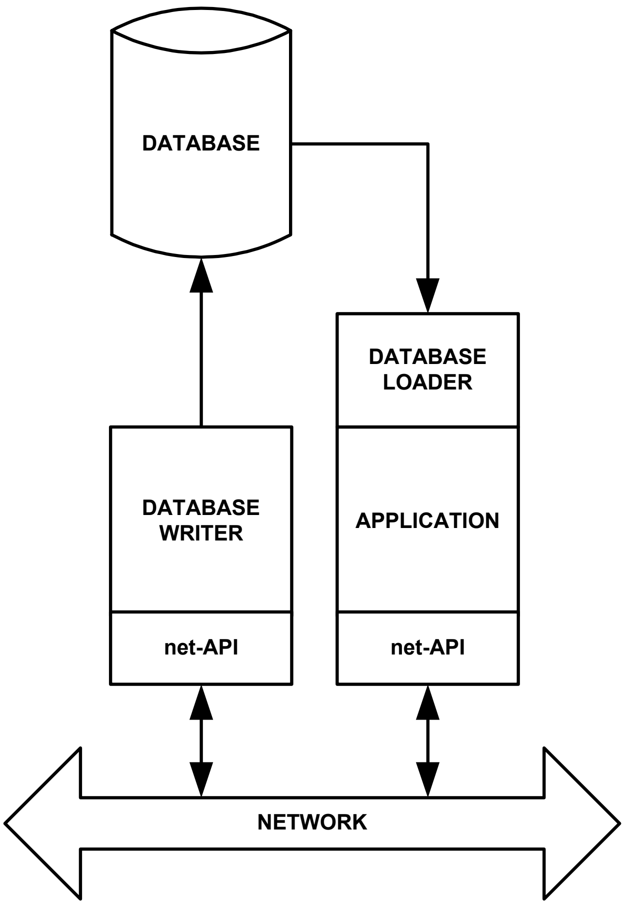
```
void CAuthor::Copy(const CAuthor& newobj)
{
  m_nID = newobj.m_nID ;
  m_strFirstName = newobj.m_strFirstName;
  m_strLastName = newobj.m_strLastName;
  m_strPhone = newobj.m_strPhone;
  /*
   * and so on, <skipped>
   */
}
```

This is represented pictorially in figure 1. As far as you might guess, these cpp-objects (like **`CAuthor`**) should not only be sent and received through the network, they also to be kept in a database. For this purpose there is a special application - “database writer”, and as expected this is loaded from

GEORGE SHAGOV

George was born in Moscow in the seventies and graduated from Moscow State Technical University. He started as a software developer in 1992 and since has spent a year and a half in the US (also developing software). George is currently working as a software engineer at Deutche Bank in Moscow. You can contact George at george.shagov@db.com

Figure 1



companies live like that, big ones, enterprises, for years

database, so there is additional functionality needed that loads these objects from the database. We have some kind of database vendor, which requires its own structures for reading and writing to the database to be filled out. This database object looks like this:

```
class CDBAuthor
{
public:
  db_int ID;
  db_string<32> FIRST_NAME;
  db_string<32> LAST_NAME;
  db_string<16> PHONE;
  /*
   *and so on, <skipped>
   */

public:
  BEGIN_DB_ENTRIES(CDBAuthor)
    DB_ENTRY_INT(ID, READ_WRITE);
    DB_ENTRY_STRING(FIRST_NAME, READ_WRITE)
    DB_ENTRY_STRING(LAST_NAME, READ_WRITE)
    DB_ENTRY_STRING(PHONE, READ_WRITE)
    /*
     *and so on, <skipped>
     */
  END_DB_ENTRIES();
};
```

As a consequence of this, there should be some functions that convert `CAuthor` to `CDBAuthor`:

```
void ConvertToDB(CDBAuthor& dbAuthor,
                 const CAuthor& author)
{
  dbAuthor.ID = author.m_nID;
  dbAuthor.FIRST_NAME  = author.m_strFirstName;
  dbAuthor.LAST_NAME  = author.m_strLastName;
  dbAuthor.PHONE = author.m_strPhone;
  /*
   * and so on, <skipped>
   */
};

void ConvertFromDB(CAuthor& author,
                   const CDBAuthor& dbAuthor)
{
  author.m_nID = dbAuthor.ID;
  author.m_strFirstName = dbAuthor.FIRST_NAME;
  author.m_strLastName = dbAuthor.LAST_NAME;
  author.m_strPhone = dbAuthor.PHONE;
  /*
   * and so on, <skipped>
   */
}
```

And of course `select` and `insert` statements are needed and look like this:

```
static const std::string s_strSelectAuthor =
" select  "
"    ID "
"    ,FIRST_NAME "
"    ,LAST_NAME "
"    ,PHONE"
" from tAuthor"
;

static const std::string s_strInsertAuthor =
" insert into table tAuthor ("
"    ID "
"    ,FIRST_NAME "
"    ,LAST_NAME "
"    ,PHONE"
") values ("
```

There should be a procedure which binds the variable to the `insert` statement. It is good if this procedure is created according to the data from the `BEGIN_DB_ENTRIES` macro (it would be a helpful automation.)

Still here? Good. Let me conclude a preamble a little bit. If you have a task of adding one field you need to:

- add the field to the class
- do not forget modify Copy function
- add the field to db class
- modify two conversational functions:
- from database.
- to database.
- modify select and insert statements

All these operations are performed manually, including compiling/testing of all the code modified. You're having a busy day! I am not joking - it is a sample from real life, the companies live like that, big ones, enterprises, for years.

But now is that all? Almost.

Figure 2

```
# Comment line
#cpp name   | aapi field | type   | dimension | DB field
| ID        | id         | int    | 1         | ID
| FirstName | firstname  | string | 32        | FIRST_NAME
| LastName  | lastname   | string | 32        | LAST_NAME
| Phone     | phone      | string | 16        | PHONE
```

Make life easier, parse it!

Let us consider a plain text file called `author.txt`, with content as in figure 2.

Having been parsed, the resulting information is absolute in order to:

- generate the class (**`CAuthor`**) and all of its members (fields) with all access functions to these fields (getters and setters)
- generate functionality which will fulfill net-object (both sides, to and fro)
- generate **`Copy`** function for cpp-class
- generate database db-class (**`CDBAuthor`**) within all its fields.
- generate two conversational functions:
- converting from **`CAuthor`** to **`CDBAuthor`**
- and back
- generate SQL statements
- inserts
- and selects
- generate whatever I have forgotten to mention.

All we need to do is to parse this simple text file. Somebody might say the parsing procedure is quite complicated, in order to argue that I am putting the flex skeleton which parses the file in Listing 1.

The result of parsing is shown in figure 3.

Listing 1

```
%{
#include <stdio.h>
#include <stdlib.h>
%}

%option 8bit outfile="scanner.cpp"
prefix="test"
%option nounput nomain noyywrap
%option warn
%x COMMENT
%x CPP_NAME
%x NET_NAME
%x TYPE
%x DIMENSION
%x DB_NAME

alpha      [A-Za-z]
dig        [[:digit:]]
thename    ({alpha}|\_|{dig})*
thedigit   ({dig})*
delim      "|"
space      [ \t]
spaces     ({space})*

%%

"#" {
      BEGIN(COMMENT);
    }
{delim} {
      BEGIN(CPP_NAME);
    }
{spaces} {
  }
```

In conclusion to this part, all we need to do in order to add a new field to the system is to modify our test file (`author.txt`) and restart the building procedure. All the functionality will be automatically generated. The parsing and generating of code is the very place where all of the verifications are to be made, and perhaps some inserts at the helpful developers database, in order to keep the data handy.

Listing 1 (continued)

```
<CPP_NAME>{spaces} {
  }
<CPP_NAME>{thename} {
    printf("Field: ");
    printf("cpp-name:%s; ", yytext);
  }
<CPP_NAME>{delim} {
    BEGIN(NET_NAME);
  }
<NET_NAME>{spaces} {
  }
<NET_NAME>{thename} {
    printf("net-name:%s; ", yytext);
  }
<NET_NAME>{delim} {
    BEGIN(TYPE);
  }
<TYPE>{spaces} {
  }
<TYPE>{thename} {
    printf("type:%s; ", yytext);
  }
<TYPE>{delim} {
    BEGIN(DIMENSION);
  }
<DIMENSION>{spaces} {
  }
<DIMENSION>{thedigit} {
    printf("dim:%s; ", yytext);
  }
<DIMENSION>{delim} {
    BEGIN(DB_NAME);
  }
<DB_NAME>{spaces} {
  }
<DB_NAME>{thename} {
    printf("db-name:%s;\n", yytext);
  }
<DB_NAME>\n {
    BEGIN(INITIAL);
  }
<COMMENT>\n {
    BEGIN(INITIAL);
  }
<COMMENT>. {
  }

%%

int main(void);
int
main ()
{
  yyin = stdin;
  yyout = stdout;
  yylex();
  printf("\n\nTEST RETURNING OK.\n");
  return 0;
}
```

Figure 3

```
Field: cpp-name:ID; net-name:id; type:int; dim:1; db-name:ID;
Field: cpp-name:FirstName; net-name:firstname; type:string; dim:32; db-name:FIRST_NAME;
Field: cpp-name:LastName; net-name:lastname; type:string; dim:32; db-name:LAST_NAME;
Field: cpp-name:Phone; net-name:phone; type:string; dim:16; db-name:PHONE;

TEST RETURNING OK.
```

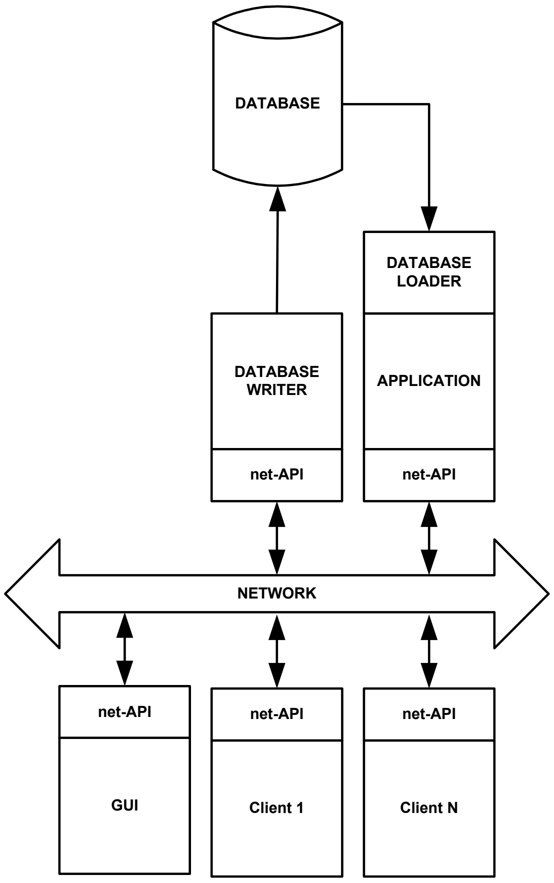
If at first you don't succeed, parse, parse and parse again!

The data, cpp-objects, are not plain, as in the sample. There is a hierarchy of them, which are sometimes pretty complicated. It means that having one 'plain text file' is not enough, we will have got a number of them, such as:

- description of net-fields
- description of database fields
- file with relations between database and net-fields
- and each particular cpp-object should have its own 'plain text file' including all of the required fields. Perhaps this is the same as a previous clause. So one item might be annihilated from the list

But for the rest of the code, all of the above is going to be generated. Someone might say the benefit is not obvious, then let us consider figure 4.

Figure 4



As you may have surmised, we were talking only about a server application, but the system itself is much more complicated. There is a GUI for the server and some additional clients, some of which are written in Java. And this net-library (or rather to say net-objects), mentioned before is (are) an interface between server and different clients. The simple question is though: how does the look up relationships between server and GUI fields work? The situation is pretty simple. Usually in GUI there are different forms of data representation (what are the GUIs are for?) or grids within the rows of data. It assumes that each particular column in the grid (or of the form) has its name. But where this name is taken from? It is neither the name of the DB field nor the name of the net-field, so usually these names are configured on the GUI side, and the relationship between these names and net-names or db-names are known only if you are taking look at the GUI. If we follow the proposed approach it means that these GUI-names are to be added at the 'plain text file' in order to have GUI-code generated also.

But now is that all? Almost.

Since the fields are going to be written in the database, it means we must know the length. There is no any reason to have dynamic strings, constructed by means of **std::string**. Therefore our cpp-class might look like this:

```
class CAuthor
{
public:
  int GetID() const { return m_nID; }
  void SetID(int nID) { m_nID = nID; }
  const char* GetFirstName() const {
     return m_strFirstName; }
  void SetFirstName(const char* strFirstName)
  {
    if (!strFirstName)
      return;
    ::strncpy(m_strFirstName, strFirstName, 32);
  }
private:
  int m_nID;
  char m_strFirstName[32+1];
  /*
   * and so on, <skipped>
   */
};
```

Why do we need that? Performance. Each particular change of **std::string** will cause a memory allocation, which is quite a costly operation from the perspective view of performance. By means of such an approach we are avoiding these additional allocations. And further, the **CAuthor** class becomes a scalar, it means that its **::Copy** member might be realized as a simple call to a **::memcpy** function.

This approach significantly boosts the performance of the system. But now is that all? Yes, it seems like, save documentation, which is realized as twiki pages and updated, of cause, manually.

Postscript

I am not going to waste your time describing the benefits of a centralised approach, as they are clean. The fashion of writing macros of declarations of static maps originates from big software companies like Microsoft, for instance, since they, distributing the technology (like ATL) are not able to use flex or yacc, since it does not look professional, if it possible to say so. Therefore many companies, are actually being quite deceived by that and are misled from the straight path, which quite disappointing. Of course what am I saying here is my own point of view on the subject.

Conclusion

In conclusion I thought to put down a couple of keen words, yet restrained myself from doing it, for who am I? I am not a judge, certainly. Moreover, during writing the article I tried to be as neutral as possible, keeping technical style, quite critical, not personal. Notwithstanding saying nothing does look strange also, therefore I am citing the words, God-a-Mercy I am satisfied withal:

"For with what judgment ye judge, ye shall be judged: and with what measure ye mete, it shall be measured to you again" Matthew 7:2 ■

Trees, Roots and Leaves #1

Örjan Westin examines the value of tree structures.

Trees are very useful; they can help with sorting and organisation of data, parsing and problem solving. They can also provide the over-worked programmer with shade and a sturdy support for the hammock. With this in mind, I was quite surprised when I couldn't see any when I went looking recently. True to the proverb, I saw a forest. Tries, binary trees, Cartesian trees, search trees... fine, but I needed a tree, and I couldn't find any.

Ah, well, I thought, I'll just have to write my own. How hard can it be? As it turns out, this was not an entirely trivial exercise, which is why I decided to write an article about it. I got a bit carried away, so it actually became three articles, roughly covering design, implementation and refactoring.

Not big and not clever

Most examples of trees I have seen have been, in one way or another, quite explicit about the difference between trees, branches and leaves. This approach has two significant drawbacks: it prohibits the use of trees and branches interchangeably in generic functions, like sorting and searching, and it prohibits the promotion from leaf to branch.

When would you want to do the latter? Well, in HTML it is quite common to see things like `<em>important</em>`. If we parsed this in a way that had each HTML element, or tag, as a node in our tree, this would be an "em" leaf with "important" as content. But if we wanted even more emphasis, and put `<em><strong>important</strong></em>` in our tree, then the "em" element would be a branch with a "strong" element as a child.

I also found that many of the trees out there have either very specialised or very big interfaces. They tend to assume a lot about how and for what they are going to be used. While I do the same, to some extent, I have made an effort to make it as simple and generic as possible.

The following requirements summarise my needs:

1. Data indifference, i.e. generic and able to hold different data types in different instances.
2. Small, simple interface, with as high exception safety as possible..
3. Whole-part indifference, i.e. offering the ability to treat roots, branches and leaves alike.
4. Composable and decomposable, i.e. it should be easy, fast and cheap to merge and break up.
5. Unsorted but sortable, i.e. a client should be able to sort a tree, but the tree should not make any assumptions.
6. If possible, conform to STL container requirements.
7. Faithful, i.e. with a strict control of relationships.

These requirements affect the design in a couple of surprising ways. The first is a no-brainer: make a template class. I already knew that, but it's always a good idea to put down the requirements you take for granted – that makes it easier to question them later, should you need to. This also goes for the second, which is a requirement that should be in place for almost any class you'd care to design – the good old KISS principle, here in the alternative meaning "Keep It Simple and Safe".

ÖRJAN WESTIN

Since 1993, Örjan has worked as a developer and consultant (apart from a return to Mid Sweden University in 1995 where he created and taught a course in "GUI design and Windows Programming"), mainly using C++. He is currently working for Unilog Ltd as a Senior Consultant and can be contacted at orjan.westin@unilog.co.uk

The third and fourth tells me I am not actually looking for a tree, but a tree node. Whether it is a root, branch or leaf is only determined by its relationships: a root has no parent, a leaf has no children, but a branch has both. If they are all of the same class, and can be treated uniformly, it helps a lot with the fourth requirement while also making the interface smaller.

the user might be surprised by the result of the assignment, which is rarely a good thing

The fifth asks for iterator support, to allow the use of the standard sorting algorithms. How to sort the tree is not my worry; as long as you can go through a list of branches with a common parent and swap them around, the client can implement sorting. The simplest way of achieving this is to make it appear like one of the standard containers, as wished for in the sixth requirement.

While full conformity with the STL containers is nice, as it makes the class easier to use, it's only an absolute requirement for library writers. For the rest of us, it's good enough if it does the job in a reliable and efficient manner. For this reason, I will do this backwards, i.e. start making a usable class, while keeping the standard in mind, and refactor it once I have it working. Normally, the arguments for this would be a lot weaker – one should always strive for standardised and familiar interfaces where applicable – but in this instance, I already know that it is impossible to make this fully conformant. But I'll tell that story later.

Finally, the seventh prohibits assignment and copy construction. This was a bit of a surprise, but it makes sense if you think about it. When I copy a leaf or branch – do I give the copy the same parent as the original? If I do, the copying of one node will alter another, since the parent must be notified it has a new child. If I don't, the copy will not be true, and I cannot do the same thing with the copy as I can with the original – in this case accessing the parent – which is hardly what you'd expect of a copy. It would definitely not be a faithful copy.

Without knowing how it's going to be used in every instance, I can't in good conscience decide which path to follow. Either way, the user might be surprised by the result of the assignment, which is rarely a good thing. Rather than baffle, I decided to provide a copy function that copied everything except the parent.

To be? No, it's not to be

The first draft looked something like this:

```
template <typename T>
class tree_node
{
public:
  ...
  tree_node<T> & front();
  ...
```

Right, I'll stop there because I have already made hamburgers of two holy cows, and I can see the friendly neighbourhood mob approaching with their quaint pitchforks and torches. I hope they're bringing something to drink, too.

The first is seen in the return type of `front()`, which is a node, rather than the template type which is the norm in the standard template library. When a client calls `front()`, the expected return is a reference to the first element held – that's how `deque`, `list` and `vector` works. Well, that's what you get here as well.

patterns are only useful idioms – you are not required to follow them when they don't make sense

Unlike those containers, however, the `tree_node` is not one-dimensional. In a list, for instance, you get hold of the first value and use it, and what you use it for is wholly dependent on the type of element you have in your list. This would only make sense in a `tree_node` if it is a leaf; if it is a branch or root, you might want it for its value or you might want it for its children. Rather than try to provide two different idioms, I stick with one.

Remember the third requirement? Only by admitting that a `tree_node` contains `tree_node`s can I make it whole-part indifferent. Everybody knows that a `list` contains `structs` holding not only data, but also pointers forwards and backwards, but this is an implementation detail and can be ignored. In this case, however, it's a design feature, not an implementation detail, and as such it should be visible, and should generate a compilation error if used wrong:

When it comes to class purposes, there can be only one

```
tree_node<string> tree("Hello");
string s = tree.front();
// Error: no conversion from tree_node<string>
// to string
```

The second holy cow on the barbecue is the composite pattern, in which, as we all know, the composite class is derived from the stand-alone class.[1] So why don't I follow that pattern?

First of all, patterns are only useful idioms – you are not required to follow them when they don't make sense. Secondly, the composite pattern is not this generic; like all good patterns it is quite specific when it comes to circumstances it is applicable. In this case, you write both the base and composite classes, so you know what you are doing. That luxury is not available when inheriting from an unknown type. Thirdly, in the words of Herb Sutter: "Only use public inheritance to model true IS-A, as per the Liskov Substitution Principle (LSP)".[2]

You cannot argue that a tree to store, say, Widgets is-a Widget, as it has no knowledge of what a Widget is. Instead, you end up with a class with two distinct and unrelated purposes. When it comes to class purposes, there can be only one. Okay, so nobody will come and cut your head off with a sword if you write a class with two or more different purposes, but I am sure I'm not the only one who have wished for that when coming across a piece of old code to maintain.

Short of actual beheading, I know there is no way to stop some people from ignoring issues of style and try to resurrect both cows by re-writing the class declaration like this:

```
template <typename T>
class tree_node : public T
{
  ...
  tree_node<T> & front();
  ...
```

Neat, or what? This follows the pattern, and it brings `tree_node::front()` back on track, so that it returns a reference to the template class. Well, it returns a reference to a class derived from it, which should be close enough. Shouldn't it? Well, as it turns out, it isn't.

```
list<string> l;
// Add elements
// ...
// Change the first
l.front() = "bubble";    // fine

tree_node<string> t;
// Add elements
// ...
// Change the first
t.front() = "bobble";    // Error: no operator=
                         // defined for tree_node
```

Oops.

And that's just the start. Even if I did not hide the `copy` constructor and assignment operator, I would hide the function front and any other functions in the template class that happens to share name with the functions in `tree_node`. Hopefully, if issues of style does not prevail, the practical considerations should convince everyone that in this particular forest, those two cows are not holy.

You want fries with those burgers?

To have or to hold

With those controversies out of the way, I can go on with the design and put the member data in. That's fairly straightforward – simple containment should do it. This means, of course, that a root or branch which is only

Listing 1

```
template <typename T >
class tree_node
{
public:
  typedef T& value_reference;

  tree_node():
    value_(NULL);

  tree_node( const value_reference value ):
    value_( new T( value ) );

  ~tree_node()
  {
    delete value_;
  }

  bool has_value() const
  {
    return (NULL != value_ );
  }

  value_reference value() {
  if ( !has_value() )
    throw std::exception("No value");
  return *value_;
 }

void assign_value(const value_reference value)
 {
   if ( has_value()_)
       delete value_;
       value_= new T(value);
 }
  ...
  private:
  T* value_;
};
```

used to keep track of children – like you would use a list or vector, for instance – always has space set aside for data, even if it's not used. If the contained class is expensive to construct, this is a bad thing.

this container is designed for recursion, something that makes it a different breed

To allow data-less nodes, I might as well keep the data as a pointer, with the appropriate care in constructor, destructor and assignment. This is the familiar pimpl idiom [3], a common an implementation of the bridge pattern[1] – see Listing 1.

Since there is no guarantee there'll be a value to return, I'll have to check for that and throw an exception if the client has asked for something I can't give. Since I'll have to check it anyway, I might as well give the client the ability to do the check, to avoid having to put a try-catch block around every call to **`value()`**. I'll also provide a **`const`** version, but there's not much point in listing it here.

That's the first bit of data sorted, what's next? The parent of course – as most people know, you have to have a parent (or two, but bear with me) before you can get children. And because most people don't want the parent to move in with them, I just keep track of its address so I can send Christmas cards. Just remember to use the right type – the parent is not the template type T, but the **`tree_node`** that has us as a child (see listing 2).

Listing 2

```
  ...
  typedef tree_node<T>* pointer;
  pointer parent()
  {
     return parent_;
  }

  pointer root()
  {
    pointer tmp = this;
    while ( NULL == tmp->parent() )
      tmp = tmp->parent();
    return tmp;
  }
  ...
private:
  pointer parent_;
```

What, no assignment of parent? No, you don't get to choose your parents, but more on that later. How do you like the function to find the root? There's another way of doing this that doesn't use a temporary variable, but recursion. If you want to, you can write something like this instead:

```
pointer root()
{
  if ( NULL == parent_ )
    return parent_->root();
  return this;
}
```

This is slightly less efficient, but illustrates that this container is designed for recursion, something that makes it a different breed than the usual two types that are distributed in the standard template library, namely: sequence containers (**`deque`**, **`list`** and **`vector`**) and sorted associative containers (**`map`**, **`set`** and their multi counterparts. We will have reason to come back to this in the next part, when we begin to play with children. ■

References

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: "Design Patterns", Addison-Wesley 1994.
2. Herb Sutter: "Uses and Abuses of Inheritance, Part 2", C++ Report, January 1999.
3. Herb Sutter: "Pimpls - Beauty Marks You Can Depend On", C++ Report, 10(5), May 1998.
4. Herb Sutter: "The Joy of Pimpls (or, More About the Compiler-Firewall Idiom)", C++ Report, 10(7), July/August 1998.

Magazine Redesign

Pete Goodliffe

As you'll undoubtedly have noticed, this issue sees a new look for C Vu. It's bolder, brighter, and hopefully much easier to read. A facelift for Overload will follow shortly.

You'll also have noticed that this is more than just a new look. We've taken a fresh view of C Vu, updated, and restructured the contents greatly. We're aiming to make C Vu more accessible, more readable, and something that will attract a new readership. This is something that better reflects the ACCU, what we do, and who we are.

I hope you find this issue of C Vu a vast improvement, and enjoy the new look. I'd welcome your comments (pete@cthree.org).

Thanks are due to all involved for their hard work on this – especially to our production editor, Alison, who has had to put up with a lot of my mithering and requests to move blocks 2mm to the left (and back again) over the last few weeks.

Don't forget to continue writing for C Vu. The magazine design is new but, as ever, the contents are entirely up to you!

Francis' Scribbles

Francis Glassborow sets another cryptic puzzle.

The unusually large number of submissions to the 'Cryptic clue' item from the last issue (note that a prize is not guaranteed, that depends on the number and quality of submissions) means I have only limited space for this column (and I have now used four lines of it).

Once again, there has been a lot of debate on comp.std.c about dangerous functions in the Standard C Library. There are a number of functions that open up the opportunity for buffer overrun problems in your code. However among all the functions in the Standard C Library, one stands out as exceptional in that it is effectively impossible to use it safely. I refer to **`gets()`**.

Apparently, it was originally included because it was in widespread use. That is not a good reason for including a clearly defective function in a Standard Library. The apparent endorsement of an always dangerous function seems like allowing an electric wiring standard to support 'diagonal' buried wires. Hiding a 'diagonal' wire with plaster is always dangerous. It being common practice among cowboy electricians would not justify endorsing it in a professional standard. We all know that removing it from the Standard would not stop it being used but nonetheless removing it immediately might make the point about its dangerous nature.

What would you think about a proposal to remove **`gets()`** immediately from the Standard C Library?

Commentary on problem 23/24

Have a look at the following code and comment on possible surprises.

```
#include <iostream>
struct X {
  int i;
  X(){}
};
struct Y: struct X{
  int j;
  Y(): X(), j() {}
};
Y y = Y();
int main(){
  std::cout << y.i << std::endl;
  return 1;
}
```

From Balog Pal (pasa@lib.hu)

Hm, I saw the problem in the last issue it looked so obvious not worth actual commenting...[It would be a very rare problem from me that was obvious to everyone. Francis]

In struct **`X`** we have member **`i`** that is not initialised in the constructor. That is, in itself just a moral problem – the programmer will get away with it as long as he remembers to never use **`i`** before assigning some value to it. **`struct Y`** just inherits from **`X`**, not touching **`i`** in any way. Then we have:

```
Y y = Y();
```

That is copy-initialisation, with the effect same as writing:

```
Y y(Y());
```

The compiler will create a temporary **`Y`**, invoke **`Y::Y()`**, then call the auto-generated copy-constructor with that temporary in order to construct **`y`**. The auto-generated copy-constructor will do member-wise construction like this:

```
Y::Y(const Y& that): X(that), j(that.j) {}
```

That, in turn uses auto-generated copy-constructor of **`X`** that looks like

```
X::X(const X& that): i(that.i) {}
```

And getting here we have a problem: undefined behaviour reading from an uninitialised **`that.i`**.

We could have avoided that with direct-initialisation of **`y`**.

In **`main()`** we read **`y.i`** – that would be undefined behaviour with the 'fixed' version. But if we have some magic wand that clears undefined behaviour status at **`{`** of **`main`**, with the code as it stands reading **`y.i`** is OK.

In practice most compilers will probably optimise the copy construction to do nothing. It is allowed by the as-if rule and cannot break a program that is not already broken (as ours is). Then access **`y.i`** will read the garbage, and on most real machines in the world (where **`int`** uses all the bits and have no trap values) will just output a random value and exit with 1. That's the most probable of the things that could happen, but that's no excuse for the programmer omitting that **`x(0)`** in **`X::X()`**.

Thank you for an excellent critique. The above is another reason for not inheriting from a class that has not been designed with inheritance in mind. Note that it would normally be an error to implement a base class (i.e. one designed for inheritance) with any constructor that did not fully initialise the base class data members.

Problem 25

Here is a problem from SQL for all those readers who have asked for some variety (and I would welcome 'gotcha' type problems in other languages such as Java, C# or Python as well as from C and C++)

Here is my problem (assumes at least Oracle 9i):

```
INSERT INTO my_table VALUES ((SELECT
my_sequence.NEXTVAL FROM DUAL), ...);
```

The intent is to insert a non-repeating auto-generated value into a primary key field of a table. The problem is that this method is simply naïve.

From William Fishburne <bfishburne@gmail.com>
Please explain why the method is naïve. Submissions by 1st May.

Cryptic clues for numbers

Last issue's clue:

On reflection, this issue is still the same prime. (3 digits)

Very curious, why did I have more answers to this clue than for all those in the previous six issues?

- From: Roy Read <roy@camelot.demon.co.uk>
  Once again I am either going to make a monumental fool of myself and give a daft answer or else you have made this one far to easy.

  Primarily another year's series begins.

  Thank you for publishing the answer to last issue. Sadly the old brain is slowly going – in my "mind's eye" I could see the boat, could hear the signature tune, even picture the face of the lead actor but could I not think of Hawaii 50 ! expletive deleted.
- From: Richard Shaw <ashawthing11@ntlworld.com>
  One too many for even 'The Power' to get in 3.

continued on page 28

FRANCIS GLASSBOROW

Francis is a freelance computer consultant and long-term member of BSI language panels for C, C++, Java and C#. He is the author of 'You can do it!', an introduction to programming for novices. Contact Francis at francis@ronbinton.demon.co.uk

Standards Report

Lois Goldthwaite brings us up-to-date with the latest news.

The standards column in the last issue of CVu discussed the fast-track ballot on C++/CLI (*aka C++ for .NET*) and some of the concerns which the BSI C++ panel has about approving this as an ISO/IEC standard on a par with 14882, the C++ Standard.

The UK put forward a procedural objection to this ballot, and since then we've been surprised to see that some websites, and even some journalists, have considered this newsworthy outside the ISO community. (No journalist has contacted me, but Herb Sutter says he has received calls.)

These other people have laid emphasis on our 'demand' for a change of name for this new language. What our paper actually requests is that ECMA withdraw the fast-track proposal, with changing the name a secondary option if they feel they must have ISO approval for their standard. ECMA are a recognised standards body and have already adopted C++/CLI as an official ECMA standard, and that will remain true whatever happens in ISO.

There are arguments both for and against changing the name. The first point to consider is that about half the people in this world who consider themselves C++ programmers are writing code for Windows and compiling it with Microsoft tools*. If Microsoft's compiler suddenly changed its name to Visual Something-Else, then about half the industrial uses of C++ would disappear at a stroke.

Some people say: *So what? If they're not writing proper C++, they shouldn't be allowed to delude themselves that they are.*

Other people take the position: *No one who aspires to understand C++ is beyond salvation; all it takes is missionary work to bring them to enlightenment. Don't arbitrarily cast them into the darkness away from C++, to wander forever in the wilderness.*

Another consideration is the extensive work that TG5 have devoted to ensuring that standard-conforming C++ code will still have the same semantics in the CLI, or .Net, environment. Changing the name might remove some incentive for Microsoft and TG5 to maintain this compatibility.

Opinions on the BSI panel are still divided about whether a radical change of name is the best course, but we all agree that 'C++/CLI' is all too easily shortened into just 'C++', leading to blurring and confusion. Indeed, Microsoft's policy was explained to me by one of their representatives: *Guidance for usage of the term "C++" in Microsoft material says that we should refer to "Standard C++" if talking about the ISO standard. If the article is about C++/CLI, it should mention "C++/CLI" up front, and then use "C++" for the remainder of the article.*

A glance at the MSDN online documentation suggests the latter rule is often overlooked in practice. Microsoft's documentation is often referred to by programmers writing for other environments, who might not realise when the discussion strays away from standard constructs and into platform-specific extensions.

TG5 should be recognised and applauded for their achievement in making C++ a first-class language in the CLI world. Many of the fundamental design principles of CLI are contrary to the fundamental principles of C++, so this was not a simple project.

There are actually three levels in C++/CLI that can be distinguished:

1. support for standard C++ code to run in the CLI environment.
2. some minor extensions to allow C++ code to consume CLI components authored in other languages, which are no more radical than any other compiler's platform-specific extensions.
3. some major extensions to enable CLI components to be written in 'C++' for consumption by other programs.

The final group are the parts which arguably should be given some other name to prevent confusion. Visual C++ can be used to compile standard C code, with the right switches, and if it also compiles this other language in addition to C++, that would be useful, too.

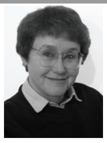
As for a name for the new parts, maybe 'CLIC++', without the slash, would be less prone to being shortened; or perhaps a better name reflecting the language's heritage can be found. Suggestions on an electronic postcard to standards@accu.org are welcome.

In my opinion, the very worst thing that could happen would be for all these extensions to become identified in the public mind with 'C++', and for 'Standard C++' to take on the connotations associated with 'Standard Basic' or 'Standard Pascal' – ISO standards for dead languages produced long ago by now-defunct working groups.

*The second point to consider is the well-known fact that 69.4% of all statistics are made up on the spot!

LOIS GOLDTHWAITE

Lois has been a professional programmer for over 20 years. She is convenor of the C++ and Posix standards panels at BSI. Lois can be contacted at standards@accu.org.uk



Francis' Scribbles (continued)

- From: Alan Stokes <alan@alanstokes.org.uk>
  Controls, aha! We need a U-turn, and a degree more.
- From: Mark Holloway <hollowaymr@hotmail.com>
  One after before hexadecimal (hex 0xB4 = 180 decimal, one after which is... 181!)

  I hope I have understood the point of the puzzle. I was briefly trying to get something with word play: Self-cannibalism, as Prince Charles might say. The clue hopefully leading to "one ate one" but I suspect that's (i) too cryptic, (ii) not numeric and (iii) just odd.
- From: richard.brookfield@bigfoot.com
  Despite adding one to a straight line, it sounds like I ate one.
  Darts player threw one over maximum; it sounds like I ate one.
- From: Dr Boots <drboots@galactichq.org>
  I ate one in my prime.
- From: simon.cornell@uk.bnpparibas.com
  Get to the root of a pair of bytes.
- From: Raymond Butler <butlerra@lsbu.ac.uk>
  Old enough to vote, one follows the majority in little more than a U-turn.

I added a little polish to a couple of these. However I kept my fingers off the winner. If Mark likes to email me, we can discuss a suitable book prize. I actually made the choice more for the 'self-cannibalism' idea than for the submitted clue.

This issue's clue

'All for one', no, 'Musketeers for musketeers'. You will need some powerful luck to get this one.

What is your alternative clue for the same number? (Submissions by 1st May). ■

Mailbox @ C Vu

Your letters and opinions.

The debate over debuggers doesn't seem to be dying down much. Bill Rubin in the USA had this to say.

The term "debugger" is a misnomer. The tool we call a "debugger" does not debug code any more than a design tool does design. Humans do these things. We would never call a design tool a "designer".

To read the commentary on the Undo article (Volume 18, No. 1), one would think that the only purpose of a debugger is to help diagnose bugs. But this is not the case. A debugging tool is just as useful in helping a developer to understand how a program works, and from a very different viewpoint than can be obtained in any other way. One might say, with apologies to the "intelligent design" folks, that program documentation – or even source code – is "just a theory". The debugger tells you what's really happening.

For example, suppose you need to add some functionality to a system of a hundred-thousand lines of code. Even if the code base is the best-written and best-documented in the world, there is no substitute for being able to step through the relevant parts with a "debugger", to get a feel for how it really works. And if the system is less than perfect, or its documentation is less than complete – or less than correct – the debugger is an essential tool, even if you never use it to diagnose bugs.

In my view, much of the "debuggers considered harmful" controversy goes away once one distinguishes between the tool and the various activities it supports, only one of which is debugging. A pity it's too late to give the "debugger" an apt name.

Meanwhile, the fun and games over null references (SCC, 17:6) carries on. Rahman Fazl had this to say.

To my mind the key point in the quote from the standard that I referred to (8.3.2/4), is that creating a 'null' reference requires binding a reference to the "object" obtained when you dereference a null pointer, which causes undefined behaviour. (For "undefined behaviour" you can read "core dump".)

So, in the example provided by Jim Hyslop, the code does not create a null reference, though it does try (at runtime) and coredumps, if a decent compiler was used.

This distinction might be considered pedantic by some. However, when giving advice to less-experienced programmers (which I think is part of the rationale behind the SCC) I do feel it is paramount to be accurate.

Perhaps what I'm trying to say is: Instead of telling people not to create null references, it is sufficient and simpler to tell them never to attempt to dereference a null pointer.

This covers a wider range of problems without introducing yet another concept into C++. I hope we all appreciate the value of not gratuitously expanding the set of concepts we need to juggle with as C++ programmers, given that the topic of C++ being seen as overly complicated was covered both in C-Vu (Lois' Standards Report) and in Overload (Editorial) last issue.

Given the changes in the education system within the UK, it seems that Francis' comments in the last issue have caused some consternation. A reader (who wished to remain anonymous) had this to say.

Normally, I read Francis' Scribbles as I find his sense of humour, while dry, always worthy of my time. However, as a teacher of many years now, I really must object to his comments in 18:1. I don't know if Francis has taught in a school, but going by his book, it would not surprise me if he had not taught in Further Education – it has the correct feel for someone experienced in that area.

I found the comments about maths (as taught) and maths teachers uninspiring to be plain wrong. While there is no doubt that throughout the 1970s and 1980s maths was very much a talk and chalk subject which required a lot of cognitive and abstract skills and was largely taught by fairly old teachers, that changed in the 1990s. Largely due to changes in curriculum, the ability for schools to "get rid of" teachers who were past their sell-by date and the advent of technology.

Maths in 2006 is not Maths as it was in 1986. Maths now is an interactive practice which utilises the likes of interactive whiteboards (students can now appreciate movement in 3D as they can see it). The use of these boards is also a positive boon in other subjects such as the sciences where the manipulation of elements, the dissection of animals and rotation of racimates not only aids in learning, but involves the students.

I will not get into the argument about examination standards as that is a different matter and one which I could talk about for hours on end.

I strongly object to the insinuation that as a profession, teaching has not progressed. It has, but not in the old "linear" style. Allow me to explain. Progression, as is understood, is a linear activity; you start at point "A" and through experience, move to point "B" and then "C" and so on. There is nothing wrong with that until you analyse what the "experience" actually was.

The experience did not include very much in self-reflection, peer review, standards reports, the student/staff experience or many of the criteria now used to ensure that some of the most rigorous standards in Europe are maintained and constantly improved upon. Schools undergo OFSTED inspections roughly every 4 years, but there is nothing to stop a snap inspection at any time and the rules for these inspections for employment are very exacting (you cannot employ anyone for 6 weeks before or after the inspection).

Experience in the old "linear" style meant exactly that – number of hours under the belt. As the curriculum did not vary that much, it was possible to qualify 1970, start teaching and not really change much of what you delivered for 15 to 20 years.

Teaching and learning in 2006 is not the beast it once was. While some decry what is happening in the education system (such as the material taught), they have very little knowledge of what is actually happening. Yes, the material has changed, but in some respects, it has become much harder than once it was. Elements of what was on the A Level syllabus in 1987 are now being taught in year 9 as they are now considered relevant and integrates far better with what is being taught in other subjects; the learning experience is now more gestalt rather than the old "just accept it". Some elements have also been dropped and some that have should not have been dropped (the over-reliance on the calculator is not good!).

To sum up. I think Francis needs to go back to school to see what is really happening – he will be amazed at the positive changes. This sort of "decry the teacher" does not do him justice. I will continue to read his column, but please Francis, don't just start to spout what the Daily Mail prints.

As always, I'm more than happy to print your letters and emails. If you wish to comment on any aspect of the magazine, please feel free to. Just drop an email to cvu@accu.org. The editor reserves the right to allow a right to reply to all correspondence. ■

Student Code Critique Competition

Set and collated by Roger Orr.

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

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

Before We Start

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

Student Code Critique 38 Entries

The student wrote "I'm getting a compilation error with this program, something about the instantiation of **invArg;** I think the type checking is too strong; any suggestions?"

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

template<class T>
class invArg
{
public:
invArg(T& arg):inv(arg){}
virtual void Write()const{cout << inv << endl;}
private:
T inv;
};

template<class T>
class Exp
{
public:
T operator()(const T& base, T
exp)throw(invArg<T>)
{
if(exp<0)
1/operator()(base,exp);
else if(base==0)
throw invArg<T>(base);
else
{
T ret=1;
for(;exp--;)
base*=exp;
return ret;
}
}
};
int main()
{
for(;;)
{
try
{
long double base,exp;
cout << "Enter a base and an exponent: " <<
endl;
cin >> base >> exp;
cout << base <<"^" << exp << "=" << fixed <<
Exp<long
double>()(base,exp) << endl;
}
catch(invArg<long double>& inv)
{
inv.Write();
}
system("PAUSE");
return 0;
}
}
```

ROGER ORR

Roger Orr has been programming for 20 years, most recently in C++ and Java for various investment banks in Canary Wharf. He joined ACCU in 1999 and the BSI C++ panel in 2002. He may be contacted at rogero@howzatt.demon.co.uk

From Balog Pál

pasa@lib.hu

First I must criticize the SCC. It's not fair. And not professional. You dump a piece of code and expect comments without establishing a context. The early entries were fragments with comments on the intent. Or had enough working structure.

Suppose you faced such code on a review, would you start fixing or commenting it? If I'm forbidden to use the FUBAR stamp I would turn it anyway without further looking until I saw a clear specification on what the task was. Aren't we teaching that when the mark is unclear you drop everything and fetch someone who knows? Instead of making a couple of wild assumptions and sit down to carve the code?

Here all we have is the student's work and a question. We could answer the question that the compiler is right and the error message clearly states the problem that the **const** object presented to construct **invArg** is not good for the non-**const** reference of its constructor. And that it can be fixed by inserting **const**, sending him home. Then he'll come back saying compiler found another error. Then a warning, then it finally compiles but does not what he thought it will. Then we can say "Why, the program follows instructions, not wishes." but who will be any wiser? As those wishes are secret material like the recipe of Coke.

Another strategy is to switch to Oracle mode, and lay out a ton of 'if you wanted this, do that' stuff. Will that serve the student who must be confused enough with the fundamental issues? And burn precious resources on all the false branches of prediction? Sigh.

So we try rev-engineer the code. In main it prompts two numbers then writes a string naming them base and exponent and printing **base^exp** that the rest of the program calculates. The calculation is done by using a templated functor class with long double precision. We also see **try/catch**. I can't tell if using template, using functor, using exceptions was part of the task, and the stuff in main is just the test harness Mark I. Or the task was to produce the result. Also there's an endless **for(;;)** loop with an unavoidable return. To make sense, one of those should go.

What we don't see in **main** is any attempt to analyse the success of getting the numbers from **cin**. That is OK for throwaway test-harness, but not OK if it was part of the task.

To calculate **b^n** I'd probably think to use **exp()** in **<cmath>** as **exp(n*log(b))** but that will produce **double**, not **long double**. So a templated Exp may serve the purpose to do the calculation with different types and precision. Here we see a stateless class with a single **operator()**. I couldn't figure a good reason for that, though there may be. When I create a functor it holds some state, e.g. one of the function parameters fixed, or an accumulator for the result... Else a simple template function should to.

Let's look at that **operator()**. It takes base as **const T&** and exp as **T**, pretty suspicious. It also has an exception spec. I refer to boost and articles why we shall not use exception specs.

Then it checks for negative exponent and finding one tries a recursion. What is a good idea, but the implementation has bugs, **exp** must be negated and we're not in lisp but C++ so we shall start with keyword return.

Then we throw up if base is 0. I just can't tell why, as far as I know exponentiation is well defined for zero base, **0^n** being **1** for **n==0** and **0** for anything else. So I would definitely return that at this point if calculating the exponent was the task.

For the rest of the cases we see a look doing multiplications. The compiler gets grumpy as we try to modify base that we promised to keep **const**. Did we mess up? No, if it were not **const**, we would multiply base by **1*2*3***... And then **return ret** that still has the **1** we set before the loop. So the actual fix shall be **ret *= base;** let's re-check the loop: **ret** stays **0** if **exp** was **0**, multiplies once if 1, etc, so far so good. Negatives would cause an endless loop, but we eliminated that in the first if. What if exp is 0.5? Duh, we'll have an endless loop.

To fix that we shall look at specs again – should **exp** really be **type T** or **int** is OK? For the latter the fix is easy, just book it everywhere. Maybe it must be **type T** but restricted to integral values by a usage rule. Or not restricted, and error must be flagged. Or not restricted and value is expected. The 'you owe the oracle' register starts overloading... For the first case I'd issue this solution:

```
template<class T>
T Exp( const T& base, int exp)
{
  if(exp<0)
    return 1/Exp<T>(base, - exp);
  else if(base==0)
    return exp ? 0 : 1;
  else
  {
    T result(1);
    while(exp--)
      result *= base;
    return result;
  }
};
```

Thus we also eliminated exceptions and related stuff. But some comments on that, especially as the original question is tied to it.

The code starts with a template class named **invArg**. That makes no sense reading through. And it took me a day to figure out **invArg** probably stands for 'invalid argument'. No kidding. It has a constructor that takes a nonconst ref (for nitpickers: reference to object that's non-const (T&)). Then uses it just to initialise a value. That is definitely a no-no, we rarely take nonconst refs, and when we do it's to modify the passed object or store that nonconst ref for better times. Otherwise we take **T** or **const T&**. Fixing that removes the compiler error in question.

This **InvArg** class does nothing but stores the passed argument, and has a **Write** method that prints to **cout**. The class appears used as an exception thrown. That has a plenty of issues.

- Hardcoding stuff like **cout** in a supposedly generic-purpose class is bad. **Write()** should do write the object's state, but to the stream passed as argument. Then the user will call it with **cout**, if that is the place to write to.
- Templated exceptions smell. It is pretty hard to catch and handle them. We try to use as few exceptions classes as we can, and organize them in hierarchy so a general handler can catch them. Especially the kind that can't do anything put print a message.

So if exceptions are really needed we're better off with a single **InvalidArgumentException** class. That will hold a single string message. It can come with a set of concrete constructors or a templated one that takes the offending argument and converts it to string. (Say, using stream mechanism internally.) And a **GetMessage()** that can fetch the string. **Write()** is no longer needed, whoever wants to write out, can write the string.

In the catch handler we should write a more informative message, saying exception happened, and what it was. With careful planning on how we can continue or exit.

From Colin Hersom

colin@hedgehog.cix.co.uk

"Something about the instantiation of **invArg**?" Did you look at the message carefully? No? It is important to know exactly what the compiler says since it really is trying to tell you what is wrong, and knowing what is wrong is well over halfway to determining how to fix it. Let's put it into my compiler and see what it says:

```
In member function
   `T Exp<T>::operator()(const T&, T)
 [with T = long double]': no matching function
for call to
   `invArg<long double>::invArg(const long
double&)'
candidates are: invArg<long
double>::invArg(const invArg<long double>&)
                invArg<T>::invArg(T&)
 [with T = long double]
```

Error messages caused by problems with templates are often cryptic, but this one is not too bad. It is looking for a function with an argument type of **const long double&** but it can only find that name with a **long double&** or a **const invArg<long double>&**. The second of those is the auto-generated copy-constructor, so presumably you want to call the first. There is rarely a need for non-**const** references, especially in constructors, so does it compile if you make the argument **const**? Well, it passes that point but now there is another error:

```
assignment of read-only reference `base'
referring to the line base*=exp;
```

Hmm. What are you trying to do? No comments because you were just hacking this out? Comments help to organise your thoughts and remind you tomorrow what you were thinking of yesterday. The code seems very hard to read, did you not use a decent editor to create it? Anything half decent would at least given you some indentation so that the flow of the program was easier to follow. You can then see that the main program looks like:

```
int main()
{
  for (;;)
  {
  // work
    system("PAUSE");
    return 0;
  }
}
```

So the **`for(;;)`** is doing nothing. What is that **`system`** call doing? I can't see why you should make this little program dependent on some external file, so I think that I'll just comment it out. You could replace it by a **`getline`** or something if you want the user to respond before moving on. If you put the return statement outside the **`for`** loop then the user is prompted for another input in any case.

What else can I see? What about this line:

```
1/operator()(base, exp);
```

That statement seems stranger the more I look at it:

- There is an explicit call to **`operator()`**, which is rarely required.
- The whole statement is an expression without assignment, so its result is discarded (and the function has an indeterminate value)
- It is recursive, passing in exactly the same arguments as it was given, so if this is ever reached it will go into an infinite recursion, the best result will be a stack error. You might be lucky and not screw up your operating system. Presumably you meant to negate the **`exp`** argument.

All your template instantiations appear to need their type provided explicitly. That is not a good sign. The whole point of templates is that the compiler can deduce the type so that if, for example, you wanted to use some type like **`Complex`** instead of a **`double`** (yes, I thought you did), you only need to change the declaration of **`base`** and **`exp`** and the compiler does the rest. At the moment you have to change the call to **`Exp`** as well.

You have calls to **`Exp<type>::operator()`** but nothing else. The class has no constructor and no retained data, so why is it a class? I know that the STL has such things, but then they have a whole load of scaffolding around them to ensure that you don't have to specify the types for the template. If you want to emulate one of them, then you need to examine and understand that code. For the moment, it would seem that your example would be better coded as a template function. You can then call it like this:

```
Exp(base, exp)
```

and the compiler will deduce the argument types to be instantiated. Let's do this and see where we get to. We need to replace that **`operator()`** call as well, so that solves that worry:

```
template<class T>
T Exp(const T& base, T exp) throw(invArg<T>)
{
  if (exp<0)
    return 1/Exp(base, -exp); // Fixed this
                              // recursion
  else if (base==0)
    throw invArg<T>(base);
  else
  {
    T ret=1;
    for (; exp--;)
      base *=exp;
    return ret;
  }
}
```

Then there is that **`catch`** which also needs to know the template arguments. You made the **`Write`** function virtual to get round that? Yes, but there is not much point in a virtual function that is defined only in one class since then you can only call it from that class. If you want to be able to catch invalid arguments from a range of types then you need a base class to catch and the derived classes to throw:

```
// This is what we catch
class invalid_base
{
public:
  invalid_base(){}
virtual ~invalid_base(){}
virtual void Write() const = 0;
  };
// We throw a derived object
template<class T>
class invArg : public invalid_base
{
public:
  invArg(const T& arg): inv(arg){};
virtual  void Write() const
  {
    cout << inv << endl;
  }
private:
  T inv;
};
```

and then change the catch to:

```
catch(invalid_base &inv)
{
    inv.Write();
}
```

OK, lets show it to the compiler. The assignment error is still there. You have assigned **`ret=1`** and then not used it. Did you mean to write:

```
ret *= base;
```

That does at least get rid of the error and it compiles OK. Did you not try this code in a non-templated form first? That should have ironed out the obvious problems, and it is usually easier to test non-templated code. Shall we try it?

```
Enter a base and exponent:
2 3
2^3=8.000000
Enter a base and exponent:
```

(Looks OK)

```
3 2
3.000000^2.000000=9.000000
Enter a base and exponent:
```

(There is something odd about that output, since the first calculation gives the inputs as integers but the second one they appear as decimals. I think that your "fixed" formatting is persistent. Do you need it at all?)

```
0 0
0.000000
Enter a base and exponent:
```

(Just testing the error mechanism - it doesn't really stand out as an error, does it)

```
4 .5
^C
```

Whoops! Something doesn't like square roots. You were going to fix that later? Shouldn't you have at least put in a trap so that you know where to add the new code when you write it? That **`for`** loop looks suspicious. Why isn't it a **`while`** loop for a start? Don't you know that it is very dangerous to compare floating point numbers with exact values (zero in this case), because the binary representation might not exactly reach that value?

You can apply the integral part of the exponent by multiplication, but then you need to deal with the fractional part. The code needs to look like:

```
T ret=1;
while (exp > 0)
{
  ret *= base;
  exp = exp-1;
}
if (exp != 0)
{
  // do fractional exponent bit
}
return ret;
```

You need to look at that error reporting again. Trapping the error may not result in writing to "`cout`", indeed I would expect to write to "`cerr`" if anywhere, and you need to write something like "invalid argument" as well. However, putting the string into the class fixes it for ever, so I suggest the class writes its value to a stream provided as an argument but does nothing else. This is the minimum that this template class could do. The base has the function:

```
virtual void Write(ostream&) const = 0;
```

and the derived class overloads it:

```
virtual void Write(ostream &os) const
{
  os << inv;
}
```

then you need a utility function to pass in the stream:

```
ostream &operator << (ostream &os, const
invalid_base &err)
{
  err.Write(os);
  return os;
}
```

so the catch clause can determine the stream and everything else:

```
cout << "invalid argument: " << inv << endl;
```

I think that is enough for now. Here is the code as we have changed it, you now need to go away and work out how to fix it for fractional exponents and check that it all works for other types. Also do fix the test harness so that you don't have to type ctrl-C when you want to stop it. Oh, and look at that invalid argument `test: 0^Exp` is valid except when `Exp==0`, so maybe your error class should take two values instead of one.

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

// This is what we catch
class invalid_base
{
public:
  invalid_base(){}
virtual ~invalid_base(){}
virtual void Write(ostream&) const = 0;
};
// We throw a derived object
template<class T>
class invArg : public invalid_base
{
public:
  invArg(const T& arg): inv(arg){};
virtual void Write(ostream &os) const
{
  os << inv;
}
private:
  T inv;
};

ostream &operator << (ostream &os, const
invalid_base &err)
{
  err.Write(os);
  return os;
}

// Exp calculates base ^ exp for arbitrary types
template<class T>
T Exp(const T& base, T exp) throw(invArg<T>)
{
  if (exp<0)
    return 1/Exp(base, -exp); // b^-e == 1/b^e
  else if (base==0)
    throw invArg<T>(base);
  else
  {
    T ret=1;
    while (exp > 0)
    {
      ret *= base;
      exp = exp-1;
    }
    if (exp != 0)
    {
      // do fractional exponent bit
    }
    return ret;
  }
}

int main()
{
  for (;;)
  {
    try
    {
      long double base, exp;
      cout << "Enter a base and exponent:"
           << endl;
      cin >> base >> exp;
      cout << base << "^" << exp << "="
           << Exp(base, exp) << endl;
    }
    catch(invalid_base &inv)
    {
      cout << "invalid argument: " << inv
           << endl;
    }
    // system("PAUSE");
  }
  return 0;
}
```

From Prof. Peter Sommerlad

peter.sommerlad@hsr.ch

Firstly the original code with commentary of what I suppose to be plain wrong:

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

Source code formatting very ugly and hard to parse by humans

Use of typename in template argument lists more modern than class

Following class is used as an exception and should better be derived from **`std::exception`** or use **`std::invalid_argument`** directly.

`invArg` requires too much from **`T`** (copy constructable and **`ostream& operator<<(ostream&,const T&)`** defined

```
template<class T>
class invArg {
public:
```

Should use **`T`** or **`const T&`** as **`ctor`** type, if used at all.

A **`std::string`** for the reason would be better

```
  invArg(T& arg):inv(arg){}
```

I/O shouldn't be the competence of an exception class

```
  virtual void Write()const{
    cout << inv << endl;
  }
private:
  T inv;
};
```

Wow, so many bad things is such little code.... why a functor instead of a simple (template) function with a template function the compiler could derive the correct template instantiation without hassle.

Assumption: should have implemented exponentiation by repeated multiplication requires:

`T operator/(int,const T&) and T operator*=(T&, const T&), copy ctor.  T::operator--(int)`

```
template<class T>
class Exp {
public:
```

1. might throw different things than just **`invArg<T>`** but promises to only **`throw invArg<T>`**
2. instead of the return value, base is used as the result of exponentation, which fails because it is declared as **`const ref`**
3. parameter **`exp`** should be an integral type, otherwise algorithm fails to terminate
4. does not return a value in 2 of three cases

```
  T operator()(const T& base, T exp)
     throw(invArg<T>)
  {
    if(exp<0)
```

Might assign to **`ret`** instead of **`return`** and place the **`return`** at the end

```
      /* return */
      1/operator()(base,/*-*/exp);
```

Recursion is dangerous, endless without negation

```
    else if(base==0)
```

Guard clause is better the first case to see it more clearly

```
      throw invArg<T>(base);
```

Better; **`throw invalid_argument("Exp::operator(): base was 0");`**

```
    else {
      T ret=1;
```

Always returning 1? should be placed outside conditional

```
      for(;exp--;)
```

Does not terminate with non-integral exp value, while would be more readable

```
        base*=exp;
```

Cannot compile since base is const ref, should read: ret *= base;

```
      return ret;
    }
  }
};
int main()
{
  for(;;) {
```

Why an endless loop, when we return from within unconditionally

```
    try {
      long double base,exp;
```

As stated above, **`exp`** should be of an integral type

```
      cout << "Enter a base and an exponent:
"<< endl;
      cin >> base >> exp;
```

Fixed output doesn't work well with negative exponents, gives 0.000

```
      cout << base <<"^" << exp << "="
           << fixed
           << Exp<long double>()(base,exp)
           << endl;
    }
    catch(invArg<long double>& inv)
```

Better **`std::exception`**

```
      {
        inv.Write();
```

```
std::cout << inv.what() << std::endl;
    }
    system("PAUSE");
```

What is the reason to pause?

Might not work on non-windows/non-dos systems, non-portable

```
    return 0;
```

Why returning within the loop or at all in **`main()`**?

```
  }
}
```

The "corrected" version of mine, which still is not perfect, but at least compiling and giving some result.

```
#include <iostream>
#include <cstdlib>
#include <climits>
#include <stdexcept>
#include <boost/static_assert.hpp>
```

```
template<typename T,typename E>
T Exp(const T&base, E exp){
  BOOST_STATIC_ASSERT(
    std::numeric_limits<E>::is_integer);
    // avoids instantiation where algorithm
    // doesn't work
  if (0==base) throw std::invalid_argument
    ("Exp::operator(): base was 0");
  T ret=1;
  if (exp <0) {
    ret /= Exp(base,-exp); // a little
    // obfuscation, might use
    // ret = T(1) / Exp(base,-exp)
  } else {
    while(exp--){
      ret *= base;
    }
  }
  return ret;
}
int main()
{
  long double base;
  long exp;
  std::cout << "Enter a base and an exponent:"
            << std::endl;
  std::cin >> base >> exp;
  try {
    std::cout << base <<"^" << exp << "="
              << std::fixed << Exp(base,exp)
              << std::endl;
  }
  catch(std::exception& inv)
  {
    std::cout << inv.what() << std::endl;
  }
}
```

Commentary

I admit that the criticism from Balog is justified – I should have given some more context for the example. However I'm pleased that despite this we had three entrants for the contest. I did deliberately leave the code left justified (as supplied) rather than adding any indent because I thought it was instructive to see just how much context is lost when the formatting style is unhelpful.

I think between them that the entrants covered most of the points in the code. My remaining point is that in ISO C++ the **pow()** function has overloads for **float**, **double** and **long double** – did the student realise this? I'd like to have seen a little more discussion about why the student wanted to use a template and then simply instantiate it with long double. If a template was really required it might be sensible to rename it 'Pow' to reflect its semantics.

A subsidiary point to be aware of is that for some C++ implementations **long double** may in fact be same representation as **double**.

The Winner of SCC 38

The editor's choice is: Peter Sommerlad. Please email francis@robinton.demon.co.uk to arrange for your prize.

Student Code Critique 39

(Submissions to scc@accu.org by 1st May)

The student wrote:

"I wanted to learn how to use STL in my own code. I've got a data structure that currently has **start()** and **size()** methods returning the start address and the size of some internal data structure. So I decided to try and write it an iterator so I can use the standard algorithms.

"I'm getting a bit stuck – so I've simplified it down as much as I dare but the test code still fails to compile if I uncomment either of the lines marked 'ERR:'. Please help me get my iterator class working.

"The real class is much bigger than **tester** and doesn't use **int** but does have the **start** and **size** methods.

```
#include <algorithm>
template< typename T >
class iterator
{
public:
  // construction
  iterator(T* p) : mPtr( p ) {}
  iterator& operator=( const iterator& rhs )
  { mPtr = rhs.mPtr; return *this; }
  // Comparison
  bool operator!=( const iterator& rhs )
  { return mPtr != rhs.mPtr; } const
  // iterator operations
  T& operator*() { return *mPtr; }
  iterator operator++() { return ++mPtr; }
  iterator operator++(int) { return mPtr++; }
private:
  T* mPtr;
};
class tester
{
public:
  tester()
  { for ( int i = 0; i < size(); )
    data[ i++ ] = i; }
  int * start() { return data; }
  int size() { return thesize; }
  iterator<int> begin()
  { return iterator<int>( data ); }
  iterator<int> end()
    { return iterator<int>( data + thesize ); }
private:
  static const int thesize = 10;
  int data[ thesize ];
};
#include <iostream>
void print( const int & i )
{
  std::cout << i << std::endl;
}
void incr( int & i )
{
  i++;
}
int main()
{
  tester t;
// ERR: std::for_each( t.begin, t.end, incr );
  iterator<int> begin = t.start();
  iterator<int> end = t.start() + t.size();
  begin++;
  std::for_each( begin, end, print );
// ERR: std::for_each( begin, end, incr );
}
```

Bookcase

The latest roundup of book reviews.

In the recent past, I've had to take the somewhat unfortunate position to drop book reviews in order to keep to the page count and balance the number of articles to the number of reviews.

Not unsurprisingly, this has caused quite a bit of disquiet, so for this and the next issue, all of the books we have had to miss out will be reviewed along with new reviews.

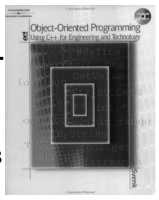
Remember, if you submit a book review you are contributing to the greater knowledge of the membership. Books are expensive and the last thing anyone wants it to spend upwards of 30 pounds on a book which is an utter turkey!

That said, if you decide to review a book, the worst that will happen is you lose a fiver – and if the book has the "Not Recommended" rating, your next book is free. What can be fairer than that.

As always, the ACCU must thank the Computer Bookshop, Blackwells and a range of other publishers for providing us with the review books.

C and C++

Object-Oriented Programming: Using C++ for Engineering



by Goran Svenk, ISBN 0-7668-3894-3, Thomson

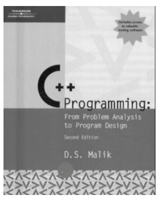
reviewed by Mark Easterbrook

Do not waste your money on this book, it does not teach Object Oriented Programming and the programming examples are a mishmash of C and poor C++. For example:

- I would expect a book on OO, when trying to illustrate code diagrammatically, to use some form of class diagram – this book uses flowcharts.
- `float` is used almost exclusively for scientific calculations, even when using library functions that operate on `double`s.
- Pre-processor macros have no place in C++ code; the author seems to like them.
- The STL is not introduced until 80% of the way through the book.

I strongly suspect the author is Fortran programmer and has just translated to C++ with the addition of OO to increase the buzzword count. Not Recommended.

C++ Programming: From Problem Analysis to Program Design



by D.S. Malik, ISBN 0-619-16042-X, Thomson

reviewed by James Roberts

When I chose this book from the 'to be reviewed' list, I was expecting a book aimed at a reader who had mastered the basics of C++ (perhaps from the same authors C++ primer book), and was interested in progressing further.

It turned out that this book is aimed at building up a student's knowledge of C++ from a start point of little or nothing.

The style is fairly wordy, and includes copious examples of completed code. Unfortunately, the author does not explain why design choices were taken, or what alternatives were not taken.

As a course book, perhaps it is reasonable to not include anything but the briefest descriptions of which compilers might be useful. However, for any other readers this is in my opinion rather important.

The main complaint I had with the book is the actual content. Why was there no mention of polymorphism (other than a passing definition)? Why were three chapters dedicated to the implementation of linked lists, queues and stacks, with no mention of the STL outside the appendices? A description of the concepts would have its place – but the full source code seems over the top.

There is apparently 'valuable testing software' included with the book. This seems to consist of a series of acrobat files mainly consisting of examination texts. I was unable to access the website, as I had no instructor id.

Developing Series 60 Applications: A Guide for Symbian OS

by Leigh Edwards et al, ISBN 0-321-22722-0, Addison-Wesley

reviewed by David Caabeiro

For those waiting for a definitive reference on Symbian C++ development for Series 60, this book fulfils all expectations. Series 60 is currently the best selling mobile platform, being deployed on devices from manufacturers such as Nokia, Siemens, Samsung, etc. It is difficult to find a topic not covered by this book, and given the lack of documentation provided by the SDK, it becomes a must-have in your bookshelf.

The book could be split into three parts. The first part comprises basic stuff such as building and deployment process, Symbian fundamental APIs and application framework (comparable to the MVC pattern). It is fundamental to understand these chapters to understand the rest of the book.

The second part refers to UI gadgets, starting with an explanation of basic controls, event handling, menus, etc. Following chapters provide description of dialogs, lists, notes, editors and many other system widgets.

Lastly, more advanced stuff, such as communications programming (sockets, TCP/IP, IrDA, Bluetooth, HTTP, messaging and telephony), multimedia framework, system engines and views, and finally testing and debugging.

Of course, no book covers all possible topics. The information you will find on some chapters (communications is an example) is the essential you will need to get started. For other advanced topics, such as client-server architecture, multithreading, etc. you will need to look for other material.

One of the things I liked most of this book is the quantity and quality of examples (which are available online) which feature working applications, so they are ready to build and run on your emulator and smartphone.

If you are on your first steps with Series 60 development get this book, you will not be disappointed. As I read somewhere, it might well be considered the "Charles Petzold" for Series 60 platform development.

Bookshops

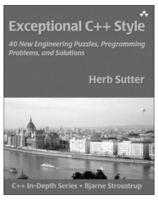
The following bookshops actively support ACCU (offering a post free service to UK members – if you ever have a problem with this, please let us know). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please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

Computer Manuals (0121 706 6000)	www.computer-manuals.co.uk
Holborn Books Ltd (020 7831 0022)	www.holbornbooks.co.uk
Blackwell's Bookshop, Oxford (01865 792792)	blackwells.extra@blackwell.co.uk

Exceptional C++ Style

by Herb Sutter, ISBN 0 201 76042 8, Addison-Wesley

reviewed by Pete Goodliffe



If you know Herb Sutter's writing then you will already be asking: is this *another* must have C++ book? Indeed it is. Herb has produced another exceptional (pun intended) tome. If you are a C++ programmer who is not familiar with Sutter's work then I suggest you get copies of Herb's previous books, work through them, and then get this one.

Sutter is a renowned C++ guru, chair of the ISO C++ standards committee, regular CUJ columnist, and conference speaker. He knows what he's talking about. As ever his latest book is well structured, readable, and authoritative.

It follows directly on from his two previous "Exceptional C++" books, and the story here is very much "business as usual". Presented in a question and answer format (which often works well, and sometimes seems very contrived), various individual topics are investigated in separate mini-articles. Some of the more thorny topics are split across several articles.

Sutter takes us on a journey through the latest wisdom on generic programming, exception safety, class design, resource management and optimisation. I was originally confused by the book's title "Exceptional C++ Style"; none of the items are really any more to do with programming style than his previous books.

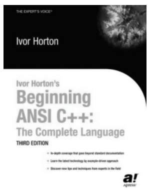
However the last section, probably the best, does finally do some justice to the title. Sutter provides a number of case studies of Real World code, showing how to improve its coding style in light of modern C++ wisdom. This section alone will help less experienced C++ programmers to learn what industrial strength C++ coding is about.

The book is well cross-referenced (internally, with his earlier books, and with other major C++ books) and clearly laid out, with sound bite "guidelines" to distil the important information. It comes highly recommended for all practising C++ programmers.

Ivor Horton's Beginning ANSI C++ 3ed

by Ivor Horton, ISBN 1-59059-227-1, Apress

reviewed by Malcolm Pell



The book's intended audience is someone with little prior programming knowledge or experience.

The book starts well, and I had no trouble understanding the basics of C++. Even though I have previous C experience, I feel that someone without C experience can still use this book to gain familiarity with C++.

The first 11 chapters cover the basics of C++, which map quite well to the features provided by C, so should not present any major difficulty to either a C user, or someone with little programming experience.

Chapters 12 to 20 cover features which are pure C++, and thus new to someone like myself coming from a C programming background.

I was surprised that 'Input and Output' is not properly discussed until chapter 19. Given that most of the example programs produce some sort of output, I would have thought that an early chapter on some basic I/O code would be beneficial to inexperienced readers.

There are plenty of sample code chunks in every chapter, and lots of useful exercises which readers are strongly encouraged to undertake. There is also a Code ZIP file that can be downloaded from the APRESS Web site.

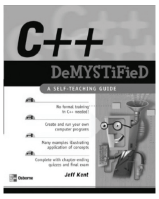
Overall, I would suggest that this book is considered by someone who desires to learn C++. Do not be put off by the number of errors found by myself and other readers. In some ways, finding these errors gave me confidence that I have understood the subject material. See the Long review for a list of errors.

[However, it seems from this review that this book introduces C++ from the traditional view of first teaching the reader 'better C'. There is nothing wrong with that approach as such, but it often fails to develop good C++ programming based on the strengths of the language. – Francis]

C++ Demystified

by Jeff Kent, ISBN 0-07-225370-3, McGraw Hill Osborne

reviewed by James Roberts



This book advertised itself as 'simple enough for a beginner, but challenging enough for an advanced student'. I would grudgingly agree on the former, but strongly disagree with the latter.

The author is strong on explanation, writing in a rather chatty and jokey style. I suppose this might be useful for readers who like the technical input leavened slightly. I found it irritating.

After a little investigation, it turned out that this was in-fact a book on C, with a little bit of C++ thrown in. For example, using `cout` rather than `printf` for terminal output. However, the word `class` only appears in a 'what to study next' chapter at the end. I think that this is a fatal weakness. I have no idea how a reader could be expected to understand a description of the `ofstream` functionality, without knowing about methods or classes.

Another criticism that I have is that the author expects the user to have Visual Studio available. Although he says that alternatives are available, it might have been nice if he had recommended one or two, or even just gave a URL for download.

Some examples will not compile, including this one that I found slightly amusing:

The following…are different in syntax, but identical in effect:

```
char name[] = {'J', 'e', 'f'.
'f', '/0'};
char name = "Jeff";
```

Yup, both fail to compile on my system too.

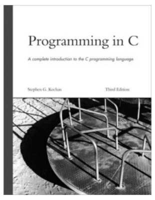
There are a number of other problems, including: no coding exercises; 'rules' which are given on one page and broken on the next; leaving function parameters as unnamed because the names 'serve no purpose'.

To end on a more positive note, there were some areas that were not badly covered (e.g. dangers of `cin` where a user might type a character when a number is expected). However, this does not make up for the inadequacies of the remainder of the book. Not recommended.

Programming in C, 3ed

by Stephen G. Kochan, ISBN 0-672-32666-3, Developer's Library

reviewed by Giles Moran



Programming in C is in its third edition and having read it I can see why it is popular. The book is aimed at the novice programmer and uses examples and exercises to reinforce each new point.

The early chapters take the reader through concepts such as compilation, variables, expressions, decision-making and loops. The layout of each chapter is clear with a number of good examples that clearly illustrate each point. Exercises for the reader close each chapter.

The reader is then introduced to arrays and functions. It is good to see concepts such as 'const' and recursion in the early chapters and not relegated to an advanced chapter located near the end of the book. C specific topics follow, with structures character strings and pointers covered in good detail. Chapters on bit operations and the preprocessor follow and these are welcome additions. Most of the introductory C books I have seen skip these features. Chapters on data types, file IO and advanced features (unions, dynamic memory allocation) ensue. The dynamic memory allocation part of the advanced features chapter should have been greatly expanded, as I do not think you can write meaningful C programs without understanding this. Four pages are obviously not enough.

The final chapter on object-oriented programming is odd, as I am reading a book on C when C++ and C# code appears. By all means, have a chapter on how to write OO C but this chapter should have been dropped and more examples on C programming paradigms could have been added.

The book concludes with a very useful appendix covering the C language and the standard library.

This is a good introduction to the C language for a beginner. It is only let down by the lack of discussion on dynamic memory allocation. Recommended for those seeking a gentle introduction to the C language.

C Programming for the Absolute Beginner

by Michael Vine, ISBN 1-931841-52-7, 240pp + CD, Premier Press

reviewed by Thomas Padron-McCarthy

This could have been a good introduction to C, but it contains many mistakes both concerning how the C language works and concerning terminology. In addition, many of the explanations are more confusing than illuminating, when they are not simply wrong.

The book starts with a short introduction on how to use Unix, and there is an appendix on how to install Cygwin to get your Windows computer to behave like Unix and to get access to the free Gnu C compiler gcc. That is only in the introduction, and in the rest of the book, the C code is not operating system specific. All examples are shown in the standard Windows command prompt.

The book matches the content of a typical beginner's course on C, up to and including arrays, structures, unions, pointers, dynamic memory allocation, separate compilation and include files. Everything is explained slowly and carefully. It stays away from more advanced topics such as varargs.

Unfortunately, there are three serious problems with this book: incorrect information about C, strange terminology and confusing explanations.

To show just one instance of incorrect C, I will quote this code example from page 60:

```
if isdigit(cResponse)
  printf("\nThank you\n");
else
  printf("\nYou did not enter a
digit\n");
```

This is the explanation that goes along with the code:

I did not evaluate the **isdigit** function to anything in the preceding **if** condition. This means that I do not need to surround my expression in parentheses. You can do this in any **if** condition, as long as the expression or function returns a true or false (Boolean) value.

Besides the author seeming to think that "evaluate" means "compare", this is of course entirely wrong. The example compiles, but only because **isdigit** is a macro which happens to contain its own parentheses (as expression macros should).

There are more errors similar to this one. Some of them, such as the claim that it is the **printf** function that parses escape sequences such as \n and \", may be explained as pedagogical simplifications. In that case it is not a good idea to lie to the student about things that will cause him to make mistakes later on.

As for strange terminology, one of the worst of many examples is that the author actually confuses variables, data types and values! For example (from page 29), he defines three floating-point variables, and says that "[t]his code has three floating-point variable data types". I do not think that he actually is confused about the things themselves, so it must be a terminology problem.

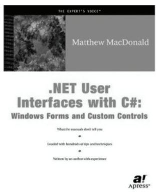
Finally, as an example of confusing explanations, consider this explanation (from page 8) of the Unix file system: *As you'll see later, directories are nothing more than files that contain the hierarchical relationship needed to support its contents and relationships.* Would you understand that sentence if you did not already know what a directory is? And, knowing what a directory is, can you decide if the statement in the sentence is true, false or meaningless?

I taught C programming for several years during the nineties, at the Linköping University and in industry, both introductory and advanced courses. As part of that work, I evaluated a large number of C textbooks. This is not the worst book I have seen, but it will require much work before it becomes a good one.

Not recommended.

C# and .NET

User Interfaces in C#: Windows Forms and Custom Controls



by Matthew MacDonald, ISBN 1-59059-045-7, Apress

reviewed by Andrew Murphy

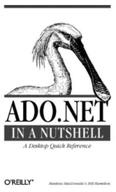
This book is great at taking MSDN, summarising it and providing a general overview. It is also easy to read and gives attention to other issues that surround a good user interface, giving you some pointers for how you should do things. However, if regurgitated MSDN offends you; you do not like lame arguments for why three-tiered architecture is bad; or you actually expected to find examples of slightly more complex subjects such as such as implementing localisation successfully, then this is not the title for you.

Chapter 1 is in my opinion the best chapter of this book. It attempts to get programmers to stop and think before they create another ill-designed "wow" interface. It tries to get us to do things in a way that users have come to expect, meaning our users can use the interface without training and with confidence. Fantastic!

On the bad side, this book is meant to be targeted at experienced C# developers so why does it spend the first half of chapter 2 trying to explain what a C# structure is? After this, most of the rest of the book is taken up with brief descriptions and MSDN method listings. Although it does give some useful tips and a CD full of examples, these examples are very simple and fall way short of being advanced.

Overall the book is easy to read and MSDN style informative but lacks substance where it is needed. At £35.00 I would not buy it but at £5.00, for reading on a train I would but I would not expect to use it as a reference book later.

ADO.NET in a Nutshell



by Bill Hamilton & Matthew MacDonald, ISBN 0-596-00361-7, 600pp + CD, O'Reilly

Reviewed by Mick Spence

This book is part of the O'Reilly "In A Nutshell" series and covers the Database API for the Microsoft .NET framework. There are three main sections, several appendices and a CD-ROM.

The first section has a brief introduction and tutorial on ADO, split into chapters each concentrating on an individual class or concept.

Although relatively new to .NET, I found this section to be readable, accurate (a few typos were spotted but nothing major), and while it did assume some previous knowledge of Databases and the .NET framework, it seemed straightforward to work through at a good pace.

The second section is a reference to the ADO core classes. Each chapter gives a technical description of one of the core classes, including a brief description of the most commonly used functions it supports.

I found this section useful and easy to read. The code examples I tried seemed to be accurate and covered the concept being demonstrated quite well.

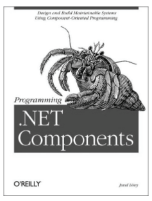
The third section offers a quick reference to the ADO API. Each chapter provides an overview of an ADO namespace with a hierarchy diagram(s), a complete list of all types defined within the namespace including a brief description of the type, and a list of all its properties / methods / etc supported.

As it says in the title, it is meant to be a quick reference, which I guess it is, but for me it appeared to be little more than a list of operations, which is a shame because the descriptions given are good.

The CD-ROM is an electronic copy of the book's third section. When installed, it integrates into the .NET IDE, allowing you to search the API Quick reference from within the IDE.

I found the book was well written, covered all aspects I expected and appeared technically accurate. The tutorial was very useful, but it takes up a large percentage of the book. The second and third sections seem more useful for all levels of experience. If you are new to ADO I would recommend this book, however if you do not need the tutorial section, you may be better looking for a pure reference book.

Programming .NET Components



by Juval Lowy, ISBN 0-596-00347-1, 458pp, O'Reilly

reviewed by Paul Usowicz

As the author himself points out the usage of the word 'component' has been slightly overdone in software. This book concentrates on developing your own components using the .net framework for component-based solutions. The book starts with very simple terminology and progresses through subjects such as ‘Interface-Based Programming’, ‘Version Control’, ‘Remoting’ and ‘Security’.

This is a very technical book which I feel should be mentioned somewhere on the cover as a novice would find this book overwhelming. The supporting web site has code downloads (for both versions of the .NET framework), an errata and other useful information. Most of the code snippets are in C# with the odd bit of Visual Basic.net.

So, did I like the book? No.

There are two reasons I really did not get on with this book. Firstly, I found it very difficult to read from cover to cover. Although I am sure many people would do this, I just could not get into the flow with this book. The chapters are too separate for any aspect of progression and I found the text, quite frankly, unexciting. Do not get me wrong, technically I was very impressed but I just struggled to read it all preferring to think of it as a reference book for specific programming tasks (a point which is enhanced by the comprehensive 35 page index).

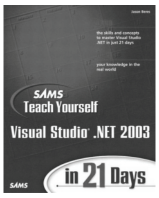
The second problem I had with this book was not so much a criticism of the book but of the fact that 90% of the information presented in this book, I already have in several others. Now I know that this is not the author’s fault but it would make the book very bad value-for-money if I was purchasing it.

In conclusion, if you are looking to do, or are already doing, component development and you do not have too many ‘professional’ books then this book is definitely worth the effort as the technical content is very good. However, if you have a huge library of good C# books then take a good look through it first to see if the component specific nature of the book is worth the asking price.

Teach Yourself Visual Studio .NET 2003

by Jason Beres, ISBN 0-672-32421-0, 666pp, SAMS

reviewed by Griff Phillips



The format of SAMS “Teach Yourself” books is highly appealing: relevant subject matter broken down into small readable chunks with exercises and review material at the end of each one. Ideal for busy professionals who need to get up to speed fast and do not have time to read the canonical texts for the given subject. Unfortunately, my experience has been that SAMS books usually fail to deliver on this promise: this title is no exception.

The main problem is that this book has no clear idea of its audience. It claims to be suitable for newcomers to Windows programming. Yet within a few pages the .NET framework is being explained in terms of COM+ and “the DNA Architecture” without any attempt to explain these terms. (However we are proudly informed that “With the introduction of Visual Studio.NET ... Microsoft has also improved the C++ language” !!) Many of the code examples are given in C#. This is an odd choice since anyone sufficiently familiar with C# to follow the examples has probably used VS.NET already and would have no need for this book. In fact the book appears to be completely aimed at .NET programmers. There is very little material for programmers who wish to upgrade from Visual Studio 6 in order to continue developing and supporting unmanaged applications.

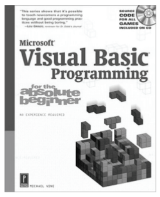
The usual SAMS sloppy editing is present throughout. For example we are reassured throughout the book that Chapter 10 will explain assemblies (a key .NET concept). In fact Chapter 10 explains ADO database access. Assemblies are never explained anywhere, nor do they even appear in the index. Throw in a pile of mistakenly repeated chunks of text, bad spelling (“lead” vs. “led” etc) and incorrectly captioned diagrams, and the end result is the usual SAMS mess.

In summary, anyone considering a book on VS.NET would be advised to avoid this one and consider spending a bit more on something from a reputable publisher such as DevelopMentor or Microsoft Press instead.

VB for the Absolute Beginner

by Michael Vine, ISBN 0-7615-3553-5, 342pp + CD, Prima Tech

reviewed by Richard Knight



This book is aimed at the novice. He uses simple games to teach the fundamentals of VB programming. The author does explain topics clearly and simply, but tends to rush over them. This style does not lend itself well to the beginner since he provides no further help or explanation.

Chapters involve programming a simple game relevant to the concepts to that chapter; source code and graphic files are all on the CD. Each chapter concludes with challenges. Like some of his explanations, no further help or answers are provided.

He devotes a chapter to debugging and error handling and emphasises the importance of checking code errors. In his introduction, he states there are no prerequisites, including knowledge of mathematics yet he fails to explain for example what a “division by zero” error is. Unfortunately, this type of omission is not uncommon.

He explains the difference between sequential and random access files but then skips very quickly over when to use which one. In one chapter he discusses in considerable depth arrays and congratulates the reader if they have understood. He concludes the arrays chapter with the comment that if you have struggled with it perhaps you are not cut out to be a programmer!

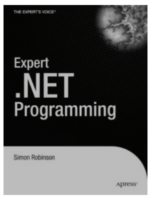
Overall I believe this book would be useful to a novice programmer but with the proviso that it is perhaps backed up with additional help, particularly form design. For anyone with any experience of VB (and that would include Access VBA) this book would prove too simple. For its price (£21.99) it would probably prove attractive to someone looking to learn VB and at that price is probably worth purchasing.

Recommended with reservations.

Expert Net 1.1 Programming

by Simon Robinson, ISBN-1-59059-222-0, Apress

Reviewed by Christer Lofving



This book is a little divergent in its plan. It is no overview of .NET, rather a series of essays covering different vital .NET topics.

The chapters (“essays”) have no immediate connection and stand all well by their own. Only exception is chapter one and two. They are connected and cover MIL, Microsoft Immediate Language, which makes up the core of .NET on this deeper level.

It’s no bold guess that many everyday .NET developers don’t even know about this assembly like, still fully graspable language. Here is a sample from the oblique “Hello World”.

```
{
  .maxstack 1
  .entrypoint

  ldstr "Hello World!";
  call void [mscorlib]
     System.Console::
     WriteLine(string)
  ret
}
```

It is possible to write a complete eShopping application out of this machine close language. But of course it would mean an enormous vast of time. Lesser known tools "hidden" in the .NET sdk are pointed out.

Among them ILDASM, the core IL compiler.

A chapter named "Going inside the CLR" is even more deep digging.

It helps to have knowledge about the MS Windows API as well as of the Java Virtual Machine, in many aspects the Java equivalence to Microsoft CLR (Common Language Runtime). The following chapter explains assemblies in depth. The inner working of this concept is valuable knowledge for any .NET developer, even a higher level one.

If you have Java/J2EE experience, you will get a lot of "flashbacks" when you read about how garbage collection is working in .NET.

Same goes for the threading chapter.

At the end the book becomes a little more pragmatic in its covering of performance, advanced Windows creation, code security and cryptography. It ends up with an extensive reference to the Microsoft Intermediate Language, dealt with in the starting chapters.

Everything is well explained and is indeed interesting reading.

But I can't help to wonder who the book is really aimed for?

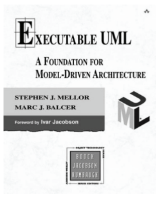
(Except for the .NET fanatical.) The philosophy of MS Visual Studio is rather to make things simple and hide complicated stuff for the ordinary developer. But, I can find out at least one more target group. Namely if you are working on any of the more extensive Microsoft certifications, for example the MCAD. As an extra reading to get a deeper understanding of .NET and to keep the interest up between exams.

UML

Executable UML - A Foundation for Model-Driven Architecture

by Stephen Mellor and Marc Balcer, ISBN 0 201 74004

Reviewed by Joe Mc Cool



Very specialised this. I can see it having both a limited market and limited appeal. As I understand it Executable UML is a further level of abstraction but resulting in some sort of executable process, some sort of "thing" I can run. I was expecting to see lots of material on comparative products, costs, ease-of-use, production environments etc.

After describing Executable UML as a "language", the authors hint at the existence of compilers, but then fall back a list of "possible" candidates:C++ variants, Java byte encoding and a UML virtual machine. So, there is nothing to play with, only more abstraction models, more state diagrams and more theory.

Overall, the book presents little added value to those already skilled in object orientated analysis of the Shlaer-Mellor type. For the perhaps naive newcomer to the world of translational methodology, the book raises a false hope for the out-of-the-box model compilation.

For academics yes, possibly; for those at the coat face, no.

To explore as background:

- http://www.martinfowler.com/ieeeSoftware/mda-thomas.pdf
- http://www.featuredrivendevelopment.com/node/572
- http://www.metamodel.com/wisme-2002/papers/graw.pdf

Real Time UML

by Bruce Powel Douglass, ISBN 0-321-16076-2, Addison Wesley

Review by Paul Thomas

First came the men in white coats flipping switches and making lights flash. Then came machine code mnemonics and code was written. Languages came next on the abstraction ladder but the text editor was still needed. The next step, we are told, is graphical editing. Throw away your vi, lines and boxes is where it's at. UML 2 is more than just an update to a syntax, it's an attempt to make it possible to write code in diagram form and it's powerful. Nowhere has this taken hold more than in the RT world where code has to be tight and coders are considered dangerous.

UML really did unify the object modelling world, we all speak a common language but sing slightly different tunes. Each camp has its own ideas about how a practitioner should take requirements through to a working system but the differences are minor. For each process, there is a tool and an endless stream of books. Usually, the books recommend using the features of UML that are implemented in the tool and gloss over the weaker. This book is one of the better ones but it only narrowly avoids being marketing literature because of it. The tool pushed is ILogix Rhapsody and most diagrams in the book are really screen-shot so you don't forget it. I can't comment on the quality of the tool, but I'm guessing that the support for communication diagrams is weak because the book devotes roughly a paragraph to them.

The first half introduces Real Time systems in general, UML 2 language features and the RT profile for UML. There's a lot to take in and it comes at you thick and fast, but it's well written and generally an enjoyable read. The small exception is the RT profile chapter that seems to be mostly lists of tags and stereotypes. A book like this could never be a comprehensive reference, so I would have preferred more space being given to tutorial. I suspect a reader unfamiliar with the basics of UML and OO modelling would struggle but everything seems to be there.

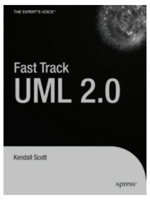
The second half of the book is given over to the more interesting subject of the software engineering process. This is the part usually skipped by trainees that just want to draw diagrams. This is a shame because analysis is where the real magic lies and where the most mistakes are made. The author makes a good job of laying out the entire process in a linear, digestible form. There's plenty of explanation of the reasoning behind the process too. I thought there was a little bit missing in the crucial analysis of use cases, but I was pleasantly surprised at the quality of the design chapters. In particular, I was impressed that patterns appear as an integral part of the design process rather than an advanced concept or, worse, an exciting new silver bullet solution.

I would recommend this book on the strength of the examples alone. There is a nice variety, they are real world rather than contrived or simplified, they are highly detailed and they are believable. If a picture is worth a thousand words, I need a bigger shelf for this book.

Fast Track UML 2.0

by Kendall Scott, ISBN 1-59059-320-0, 173 pages, APress

reviewed by Derek Graham



Kendall Scott will be well known to most people familiar with OO or UML. This latest book is a guide to version 2.0 of the UML specification. It follows very much in the style of his "UML Distilled". The difference is that that book was written at a time when UML was still new and exotic and so an element of introduction to both UML and OO was in order.

In "Fast Track" he assumes you already have some idea about the object-oriented approach and there is no introductory material. The author also assumes that the reader is someone already involved with IT and has at least a passing acquaintance with Java, web development or windows application development. Each chapter deals with an aspect of UML notation in a fairly logical order not just the conventional "one chapter per diagram" approach. Each diagram element and concept is described afresh with commentary on their use within a larger context. At the end of the book is an appendix, covering the built-in UML stereotypes, and a glossary.

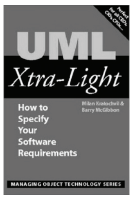
The book describes UML as it is now rather than describing how it has progressed from the earlier versions. This can make the book a little frustrating at times when you are searching for information on what the major changes are without wading through the entire text. I think the book could have benefited from an appendix drawing these changes together. The examples used throughout the book illustrate common design situations and how UML can be used to model them.

Occasional references to contemporary technologies (.Net, ATL, COM+) and software products (Rational XDE) seem to have been included as proof of how up-to-date the book is but will probably help to date it in a few years (or months) time.

UML Xtra-Light

by Milan Kratochvil & Barry McGibbon, ISBN 0 521 89242 2, CUP

reviewed by James Roberts



The idea behind this book is to give non-IT project members (business users, management etc.), enough of an understanding of UML to improve project communication via documentation. This is a laudable aim, and would be useful. However, this book does not live up to this ambition.

I would have hoped that the authors would have concentrated their efforts on giving a digestible overview of XML - possibly illustrated with meaningful examples.

There was some effort at this - but at times it seemed to take second priority to the exhortations of reuse of component based design. Whilst I would not disagree with the sentiment, it was not clear how this was appropriate in this book.

I had other grumbles with this book. Some of the discussions suddenly dropped into quite a technical level (e.g. strategies for mapping object hierarchies to database tables). All jolly interesting, but not necessarily appropriate to the target audience.

There certainly were good aspects to this book. The 'traps' section of use cases was good.

Unfortunately, although this would have been appropriate in a teaching manual for UML it was possibly not useful for the management (who might want to know how to read a use case, not to write them).

In summary, although there are good bits this book, it is not focused to a particular audience and spends too much time evangelising over the 'paradigm' of component based delivery.

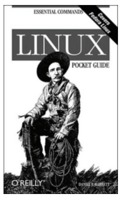
Not recommended.

Linux & Unix

Linux Pocket Guide

by Daniel J. Barrett, ISBN 0-596-00628-4, 191pp, O'Reilly

Reviewed by Ian Bruntlett



This book is very nearly an "oh wow" book. However, if you use the bash shell and could do with some gentle assistance, buy this book *now*. Unlike other books you buy, you'll get *loads* of use from this book.

At the back of the book there are 5 sides of blank paper – 7 if you disregard the sensibilities of book publishers. And if we *are* going to be utilitarian (not such a bad thing for a utility book) there are the inside front and back pages as well. I believe they should be used for categorised lists of commands you should know about – but with a note that there isn't enough space to fully detail them. man and info are your friends :)

There are the occasional errors – xclock is in the index under oclock, for instance. The surplus pages could have been put to good use as either a list of categorised gawk functions or categorised bash functions or an appendix of the commands you ought to be aware of but can't be fully documented because of lack of space.

This book is tailored to Fedora Core 1 Linux (Nov 2003) but I'm using it for SuSE Linux, Zenwalk Linux and Cygwin.

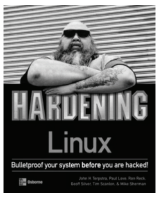
Here is a quick list of the things that *aren't* covered that I believe should be: shred, chroot, GnuPG, strace, ldd. And the section on rpm *really* should have mentioned that rpm –Va can list all the changes made to software packages since you installed them.

VERDICT – Recommended.

Hardening Linux

by Terpstra, Love, Reck and Scanlon, ISBN 0-07-225497-1, Osborne

Reviewed by Ian Bruntlett



I took this book out of the library because I wanted to introduce a Linux box to a LAN that has broadband internet access. I've got to admit the corny looking cover did put me off this book.

To coin a phrase, this is an "Oh, wow!" book. From the little experience I've got, this book covers most if not everything. It certainly covers everything I know about Linux security. And it documents commands and practices that generic Linux primers seem to ignore. That alone makes this book worth buying if you have a Linux box attached to the internet. A word of warning though – this book only covers Red Hat and SuSE Linux. I mainly use SuSE and dabble with other distros – some of the recommendations will still apply to non-Red Hat, non-SuSE Linux systems. Actually this book takes security to the extreme of describing how enterprises should handle security. It's *that* good.

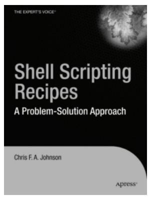
One of the key aspects of hardening a Linux box is to limit the system services running – the logic being, if it isn't running, it can't be hacked. It also recommends that compilers are kept on safe machines. If you have a machine that might be hacked, then try not to make life easier for the hackers by giving them access to development tools on a compromised system.

VERDICT: Highly Recommended.

Shell scripting recipes

by Chris F.A. Johnson, ISBN 1-59059-471-1, Apress

Reviewed by Christer Lofving



10 years ago I would have considered this book as an almost invaluable resource. (Probably unavailable as well!). Today, with the Webs enormous expansion in mind I am more doubtful. There is a good amount of free shell scripts out there. Not only for the traditional bash, but for other scripting languages like kshell and z-shell (Zsh) as well.

The latter is a pretty unknown, but very powerful one. http://zsh.dotsrc.org/ is a neat site about it which makes a good introduction for the curious. It is also in a Zsh I have tested the actual scripts in this book.

In fact, most of my shell scripting knowledge origins from online resources like free tutorials and code samples. My conclusion out of this is that nowadays a potential author of a "shell scripting recipe book" has to do a good handcraft to attract buyers. Not to say a good artwork.

Because Chris Johnson seems to have the same feeling for his scripts as I imagine an artist has for his creations. It means this book is indeed no rush job, it is very well written. Each chapter has a thematic approach, for example "Playing with files", "Scripting with numbers" and even "The dating game" (actually about date formatting)

This is nice, but makes it a bit difficult to locate "my" scripting hint. Each script in the chapters has a brief overview, an introductory "how it works" and an usage pattern like;

```
Usage: wfreq [FILE ...]
```

So far so good.

Next comes "The script" with the main body of the script, like

```
prw ${1+"$@"} |
  tr -cd 'a-zA-Z-\n' |
    sort |
      uniq -c |
      sort -n
```

A more or less cryptic "Notes" is finally ending the example up. Every script is presented in this way. It is easy to feel confused at this point.

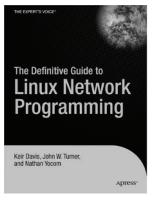
Is the code above the script itself or some kind of pseudo-code?

Nowhere in the book is there a fully complete script! A detail that is redundant for the pro, but I think of vital importance for a the lesser experienced. A very informative overview of Internet Scripting resources end up the book, along with a short but well disposed index.

The Definitive Guide to Linux Network Programming

by Keir Davis et al, ISBN 1-59059-322-7, 375pp, Apress

reviewed by Alyn Scott



I chose to review this book because it covers two subjects I am interested in, namely Linux and networks. My initial impression was that it was rather a thin book to be called a "Definitive Guide". Could it live up to my expectations in just 375 pages?

It is divided into three parts. Part one starts with the fundamentals of network protocols and socket programming. Chapter one is a concise and yet easy to read introduction to TCP/IP down to the byte level of the packet headers. The basic system functions and data structures are described as they are used in the next two chapters.

Part two is about the design and architecture of client-server programs. The different methods of handling multiple simultaneous clients are described, e.g. forking and multithreading.

Simple client-server programs are developed which leads onto developing a custom network application, a chat server. All the design decisions for a custom protocol and the implementation of the chat server and a GUI client are described in the next four chapters. There is also a section on the debugging and development cycle.

Part three deals with security. It starts by explaining tunnelling, OpenSSH and the Public Key Infrastructure (PKI). It then illustrates the many library functions in the OpenSSL toolkit with short code fragments.

Chapter 12 covers common security problems, the techniques hackers use to compromise systems and the methods we can use to make then more secure. This leads onto the second case study for a secure networked application, an authentication server. This brings together many of the concepts in the previous chapters.

Finally, the appendix gives a brief insight into IPv6 and converts a simple server and client.

Overall, this book is well structured and mostly easy to read and I could find little fault with the technical content. It is a very good introduction into network programming but I still question the use of "Definitive" in the title. There is scope to expand the later chapters with more examples as these are the most difficult to follow.

The quality of the source code was usually very good. There were a few elementary errors in some of the worked examples and the server program of the first case study would not compile.

Apart from these slight reservations, I would recommend this book to anyone interested in this area of programming

Unix & Shell Programming

by B Forouzan & R Gilberg, ISBN 0-534-39155-9, Thomson

reviewed by Paul F. Johnson

I have considered Linux and UNIX shell scripting akin to code optimisation - a black art that you either can or cannot do. I think the reason has been that all the books I have read consider the reader to be almost an expert to begin with. This book does not.

It assumes nothing other than knowing what a shell is. No requirement for being a perl guru or such. Being able to use an editor of your choice, and being able to use chmod is all that is required. Simple.

The book is very well structured with clear and concise explanations on how to write scripts. Even complex scripts are covered, when the authors patiently explain what is happening. As the reader has probably never done any form of shell scripting the authors have taken the approach of teaching a secure way and following best practice for scripts.

I really enjoyed reading this book - and I do not say that very often!

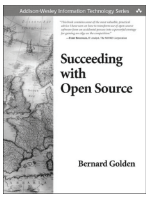
The only down point are the end of chapter review questions. While correctly pitched, sometimes the actual reason for asking a question is lost and this can be confusing. That said, I only recall seeing such questions less than 5 times in the whole of the book, so it is not a big problem.

Given the growth of Linux on the desktop (and the projected growth for 2005/6), I would suggest anyone wanting to gain employment in this ever growing sector should get this book. Study it well; it will stand you in good stead. Highly Recommended

Succeeding With Open Source

by Bernard Golden, ISBN 0-321-26853-9, 242pp, Addison-Wesley

reviewed by Mike Pentney



This book is aimed primarily at IT managers who wish to select and use open source software in their organisations. It is well written and well organised: each chapter starts with an executive summary and there are numerous marginal notes that can be skimmed to get the gist of the text very quickly. The author is CEO of a consultancy specialising in open source products and his knowledge and experience are evident from the text.

The book is divided into two major sections: the first part gives a brief overview of open source software and the second part presents a formal tool for evaluating open source products.

The overview of open source software describes what open source software is, who creates it, who uses it, where it can be obtained from and how it might benefit an organisation. It also discusses how individuals and companies are trying to profit from open source, and addresses the risks that are unique to open source products. There is a good discussion of the major open source licenses and their implications for companies wishing to incorporate open source software into their own products.

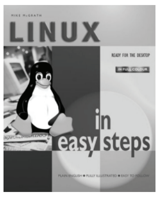
The second part of the book is a detailed presentation of a formal approach to evaluating open source products - the "Open Source Maturity Model (OSMM)". The central idea is to award points to products in six main categories: the software itself, the support options, documentation, training, integration with other products, and the availability of professional services (which could include installation, configuration, training and support.) The author uses JBoss (a Java-based application server) as an example and shows how it might score in an OSMM exercise. Although various aspects of the OSMM evaluation process are fairly obvious, and there is some overlap between categories, the model is a useful basis for comparing products and will answer many of the questions an IT manager ought to be asking before committing to an open source solution.

Recommended.

Linux in Easy Steps

by Mike McGrath, ISBN 1-84078-275-7, Computer Steps

reviewed by Paul F. Johnson



This must go down as probably the worst book I have ever read. Everything about it screams "stick with Windows". It is plain awful.

Right from the outset the books sets out on the wrong foot. It seems to think there is only one Linux distribution around (Mandrake). It does not bother with the likes of Redhat, Debian or the other flavours around. The book concentrates on dual booting rather than a fresh install – why? I can understand back in the days of RedHat 6 or 7 when hardware was not as well supported, having a dual boot enabled you to access the internet under Windows. But not for the age of this book.

While the idea of the book is commendable (easy steps, a brief look at a few applications and the such), everything is biased towards a particular distribution – the advice given for installing applications or removing them is useless for those using a non-rpm based system.

Probably the only decent parts of the book are the small sections on using OpenOffice or Gimp, but that hardly makes up for the rest of the book.

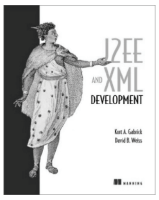
There are numerous problems with this book. Save your money. Do not buy this book – unless you want to convince someone to stick with Windows. Not recommended.

Java

J2EE and XML Development

by Kurt Gabrick & David Weiss, ISBN 1 930110 30 8, 274pp, Manning

reviewed by Alistair McDonald



This book assumes that the reader knows J2EE and XML and intends is to show how Java technologies can be used to manipulate XML. I am writing this review in 2004, and the book is a little dated, however most of the information is still current and when the authors expected major changes to occur this is clearly pointed out.

The book starts with a rather lengthy overview of distributed systems and J2EE development. The second chapter is partly an introduction to XML. The discussion quickly becomes java-centered, with the second chapter discussing all the java APIs JDOM, JAXP, etc. By this point, all the prerequisites have been covered, and chapter 3 covers the persistence of XML, and the various options open to the developer - files, databases, and so on.

The book next discusses application integration, through a web services example using SOAP. The sample code is remarkably concise, and shows what can be done with Java and XML when they are used properly. When a subject comes up, it is covered well - Web Services Description Language (WSDL) comes up in this chapter, and is covered there and then. This makes the book easier to read from cover-to-cover.

Once the main XML issues are out of the way chapter 5 covers user interfaces, using an example based on a JSP servlet using XSLT to filter the results. It is well explained and I was left feeling that developing a complex application can be very simple if the correct decisions are taken when architecting the project.

The final chapter covers a complete case study encompassing most stages of a project, and again shows the experience of the authors.

Unfortunately, book related web site appears to be neglected. I placed a comment on it, and there was no reply in nearly three weeks. I did manage to download the source, however - a whopping 2.5 megabytes in a zip file.

At some places I found the text incredibly boring and I had to shake myself and reread a paragraph. It is difficult to explain why - I appreciate the conciseness of the book, and everything said is relevant, but occasionally the writing style seems to put the reader to sleep.

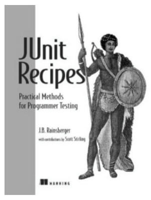
As a developer, I appreciate the book for its conciseness and breadth. Repetition is rarely apparent and the examples show exactly what is required with very little window dressing. The result is that this book is like a roadmap - it shows you where you can go with XML, giving you enough information to get there but avoiding the details, such as the exact layout of every junction. If you know Java, and you know XML but have never used the two together, or if you lack the knowledge or experience to choose an approach with authority, then this book will really help.

If you buy this book because you need to know what it tells you, then you will probably have to buy more books later. If you are not familiar with the APIs you choose, then you will need a reference manual or a guide to get the best from them. The examples in the book are concise and show you exactly what to do for the sample, but a real-life project will require more than just a few APIs.

JUnit Recipes

by J.B.Rainsberger, ISBN 1-932394-23-0, 720pp, Manning

reviewed by Anthony Williams



When I discovered an obvious naming error in one of the early examples, I was immediately concerned about the level of proofreading that was done before publication of this book. However, I am glad to say that my concerns were unnecessary; there are no mistakes of any significance anywhere in the book.

The book is laid out into a series of chapters describing related techniques, such as "working with test data" and "testing web components", with each chapter split into a series of "recipes", each describing a particular technique. The layout of each recipe is good, with a problem statement, additional background, the details of the recipe itself, and further discussion. There are also references to other related recipes that provide alternatives to, build on, or are relied on by this recipe.

The level of coverage is very comprehensive. Having read the book it feels like there is a recipe for testing just about everything you could write in Java: from simple classes; to XML generation code; database access code; EJBs; singletons, and JSPs. There are even recipes on managing your test suites, adding tests to hard-to-test classes, and the use of test implementations of other objects to allow testing objects in isolation.

Just because it focuses on Java and JUnit does not mean that this book is useless to anyone programming in another language. On the contrary, many of the techniques described can be applied in any Object Oriented language, though there are certainly some that are specific to Java. For this reason, I would highly recommend this book not just to Java programmers, but to anyone interested in improving the testing of their code.

Highly Recommended.

Learn Java in a weekend

by Joseph P Russell, ISBN 1-931841-60-8, 482pp, Premier Press

reviewed by Paul F. Johnson

Okay, I know what you are thinking – it is another one of those SAMS style "do the impossible in a really short amount of time" books. For a change, it is not and actually never claims to be. It even says to claim such a thing would be stupid!

This book is designed for real beginners – those who may have installed the Java SDK and not known what to do with it other than play a couple of online games or watch the BBC news ticker on their website.

The idea behind the book is that you have seven sessions (Friday night, three on Saturday and three on Sunday) and that by the end of the book, you should be able to write a standalone Java program or a Java applet. In the case of this book, the author uses a simple calculator program that is built up from a simple command line right up to a functioning GUI calculator. This is where things fall down though.

While the author explains what is going on, to me (and in earlier chapters especially), it appears to me that he is treating the reader as if they are bordering on slow. I can understand why he has done that, but to me (coming from a C++ background), it was not pleasant to read. Later on though, the author does start to treat the reader as a human being.

My other bind with this book is the volume of material covered. Sure the book is 468 pages, but to cover a large amount of the GUI provided by Java in a relatively short space was not a good idea and actually does an otherwise good book a disservice.

On the positive side, I have finally managed to understand the inheritance model used by Java, so the book cannot be all that bad.

Does it do what it claims to do? Yes and no. Yes, you should be able to write a standalone application or applet by the end of the book, but no as it will probably be full of holes.

Professional Java Tools for Extreme Programming

by Richard Hightower et al, ISBN 0-7645-5617-7, 732pp, Wrox

reviewed by Jim Hague

Let's start by getting the title out of the way. This book is about open source tools for Java software development projects. Yes, there is a short introductory chapter on XP, and the tools discussed may be of use to XP practitioners, professional or otherwise, but in no way do you have to be following XP or any agile methodology to benefit from using the tools described in this book.

The book is presented in the usual Wrox style, with head and shoulder shots of some of the authors gracing the front cover. It is an expanded second edition of a book from 2001, with 'Professional' tacked onto the start of the title.

Once past the introduction to XP, the book moves on via an overview of Java and J2EE deployment concepts to 3 sections covering tools for building projects (Ant, XDoclet), automated testing (JUnit, JUnitPerf, HttpUnit, JMeter, Cactus, JspTestCases, jcoverage, Jemmy, jfcUnit, Abbot) and continuous integration (CruiseControl, Anthill). Interspersed with these are chapters on CVS, Bugzilla and Maven. These are followed by 150 pages of API reference and an appendix listing the sample application used when discussing some of the tools. There is a distinct J2EE bias to the book, with much space devoted to building and testing web applications and EJBs.

With eight different authors credited, you might expect the quality of the chapters to vary, and you would be right. Several are good, particularly where the document for the tool in question is sparse. I found the chapter on "Swing testing with Jemmy" particularly useful, and the chapter on Maven a good introduction to Maven. Most of the others cover their subject in a workmanlike fashion, but I did feel that a couple offered little beyond so much rehashing of the documentation. No indication is given as to the division of authorial responsibility.

Sadly, the book is badly let down by some of the code extracts. There are obvious syntax errors, and extra spacing scattered haphazardly. Some, but as far as I can tell not all, of the code can be downloaded from the Wrox website, but even here some of the samples are incomplete.

Every Java programmer needs to know these tools exist, and the aim of the book to explain and spread good software development practice by showing how such practices can be implemented with readily available tools is laudable. This book is not a must-have, but is worth passing around colleagues new to Java. Though do warn them about the code samples first.

Real-Time Java Platform Programming

by Peter Dibble, ISBN 0 13 028261 8, 332pp, Prentice Hall

reviewed by Alan Barclay

This book promises to guide the experienced Java platform developer through everything they need to know in order to build effective Real-Time programs. In fact, I found it all quite complex and a rather boring read and did not actually feel that I had learnt enough to know how to put together a complete Real-Time program.

This is really a book for die-hard Real-Time zealots only and not for the Real-Time newcomer or casual reader thinking that they might find some Java performance optimization techniques within. It contains around a hundred pages on the general theory and problems of Real-Time systems followed by a couple of hundred pages on the solutions and techniques introduced by the Real-Time Specification for Java (RTSJ, JSR-1). These new techniques include new Non-Heap Memory, Thread Scheduling and Asynchronous Events. Finally, there are just a few pages on recommended practices.

The author does give the impression of a comprehensive knowledge of the subject and the book is well annotated with diagrams, tips and example code snippets. Unfortunately there did not seem to be may full examples of Real-Time Java in action. Instead the snippets were typically up to a dozen lines long only showing certain of the techniques in action.

As is so often the case, the stated web site containing sample code and updates is nothing but a dead link. I was unable to locate errata or source code downloads for this text.

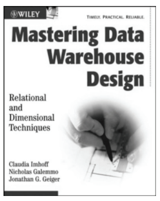
If you are looking for a guide to using Real-Time Java then, with a bit of perseverance, this book will probably help you along just fine. However you will most certainly also need “The Real-Time Specification for Java” (ISBN: 0201703238) for full details of the Real-Time API.

Database and Database Programming

Mastering Data Warehouse Design

by Claudia Imhoff et al, ISBN 0-471-32421-3, Wiley

reviewed by Richard Stones



This book tries to be a comprehensive guide to data warehousing from fundamental relational concepts (chapter 2) upwards. It also aims, according to the back cover, to provide advice on two conflicting approaches, the more traditional Inmon relational approach, and Kimball's dimensional data mart approach.

The book gets off to a rather slow start, with a somewhat unfocused chapters on basic relational concepts, business models and discovering keys, which I thought strange. If the reader needed this basic introduction, they probably are not ready to try building a data warehouse, never mind worrying about which philosophical approach was best.

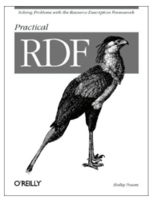
Things then improve as the book settles down into the more practical steps of getting the model correct, and deciding what data needs including. It then moves into more data warehouse type areas, looking at managing hierarchies, the calendar, and transactions, all key areas for data warehousing. Most of these chapters concentrate on relational techniques. There are some sections I personally found odd - for example some explanations of when to use bit map indexes - which seemed out of place in a book that felt it needed to start by defining normal forms and then only the first three.

By the end of the book I did not really feel the authors had addressed "head-on the challenging questions raised by Kimball" as promised on the back cover. I also found it not a particularly easy read; it never really managed to make me want to pick up the book to explore further, though to be fair I've read much worse. The book seemed strongest in the chapters when it concentrated on relational based techniques for smaller data warehouses, which are being built by people who need a refresher at the beginning of the book on normal forms. Overall a worthwhile book for beginners to data warehousing who need to build smaller relational data warehouses, but unfortunately never really justifies its "mastering" title.

Practical RDF

by Shelley Powers, ISBN 0-596-00263-7, O'Reilly

reviewed by Ivan Uemlianin



This book introduces the Resource Description Framework (RDF). It concentrates on syntax and editing tools with use cases and RDF's USPs considered secondarily. It is pleasantly written but unimaginative, adding little to what is freely available on the net. Not recommended.

After an introductory chapter, chapters 2-6 describe RDF concepts and XML syntax; chapters 7-11 describe software to manipulate RDF; chapters 12-15 describe RDF applications, including the W3C's Web Ontology Language (OWL) and RSS.

Information coverage is adequate, and the coverage of commercial RDF application is creditable, as RDF is still far from the mainstream. However the organisation of the book - syntax, tools, and applications - soon reads like a catalogue, and the reader is not drawn forwards. Some big issues hover frustratingly in the background.

First: the semantics-based nature of RDF. This is covered early on, but treated as a difficulty rather than as a fundamental property of RDF.

Initially RDF had no syntax: even now RDF/XML is one of several possible syntaxes. While SGML, HTML and XML all started as syntax and derived semantics (or a data model), RDF began as semantics, with firm mathematical/logical foundations.

The basic unit of the semantics is a proposition, analysed into subject and predicate, predicate being further analysed into property and value. This <subject property value> structure is the RDF triple. For example, the proposition 'The secretary bird has grey feathers' might become <secretary_bird feathers grey>. Given <bird_abc123 species secretary_bird>, RDF supports the deduction <bird_abc123 feathers grey>. RDF builds on this to improve the granularity of representation: for example by enabling triples to be subjects or property values (reification).

OWL provides basic datatypes like Class, ObjectProperty, DataType, upon which domain-specific ontologies can build. RDF's data structures can be tailored exactly to problem domains. The chapter on OWL is misplaced in the uses section: it is effectively part of RDF (like C++'s STL). Keeping it till the end only hides the usability of RDF.

Second: RDF and RDBMS. RDF has native support for deduction, and semantics close to natural language; it is extensible in that any two RDF data structures can easily be merged or translated into a third. These properties make RDF attractive to people dissatisfied with the relational data model: most of the commercial RDF applications covered are 'RDF databases'. The book uses the relational model to illustrate RDF concepts and gives a few urls and commercial applications, but it feels like an important chapter is missing.

It could have been better:

1 Categorise use cases (e.g., next-generation dbms, document metadata, UI design) and give concrete examples.

2 Discuss important features of RDF fully: RDF's semantic base; its support of

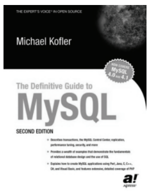
deduction; how its data model differs from other data models.

3 Review syntax and tools.

The Definitive Guide to MySQL 2ed

by Michael Kofler, ISBN 1-59059-144-5, Apress

reviewed by Christopher Hill



Riding on the coat tails of the latest trend we have yet another book on MySQL that extensively covers installation of MySQL on a number of platforms; Database design and the SQL language from a MySQL point of view and finally how to use MySQL in various programming languages.

While I get the impression that the author knows the subject well, he has not been well served by the translation, which is quaint at best and very confusing at worst – two examples “Extraordinarily frustrating was the attempt to install both packages under Windows”, and “However, if you wish to allow, for example, that in the book database a book could be entered that had no publisher, then you should do without NOT NULL”.

In Part II the author addresses database design, SQL and using PHP as the programming language; not in three separate chapters, but interspersed with much cross-referencing. I have used PHP and MySQL for a few years now, but I found the presentation very confusing. This was not helped by the code examples not matching the resulting output, or that were incomplete. I also worry about code examples that show poor practice – one example was issuing a query within a for loop to retrieve records with keys 1 to 6.

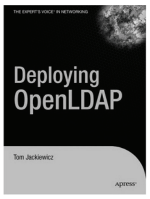
Part III romps through Perl, Java, C, C++, Visual Basic C# and ODBC. These chapters are mainly how to install the components and to get a simple “Hello MySQL” example running. These fall into the chasm of too much information for the expert and insufficient for the novice.

Not recommended.

Deploying OpenLDAP

by Tom Jackiewicz, pages 311pp, ISBN 1-59059-413-4, Apress 2005

Reviewed by Andrew Marlow



NOT RECOMMENDED

This book is NOT recommended. It is poorly presented, badly organised and lacking in coverage.

OpenLDAP is an open source implementation of some tools that use the Lightweight Directory Access Protocol (LDAP). LDAP is typically used to store directory information on email addresses but can be used for much more. The book says it is aimed at people that wish to install and configure OpenLDAP and that is where the focus of the book is. Perhaps this is why there are so many perl listings, man pages and descriptions of configuration file parameters. However, there is almost nothing on the strategic design decisions involved in deploying OpenLDAP. For example, chapter 3 "*Implementing deployment, operations and administration strategies*" seems mainly concerned with what to name the server and how to start the slapd daemon.

The book is poorly presented. The chapters are numbered but the sections and subsections are not. This is apparent from the table of contents but continues to be a distraction in the book due to the overall lack of structure. Each chapter has a summary at the end but it would be better if each chapter had an overview at the start. It appears to be quite disorganised. There are plenty of nuggets of information but one comes across them almost at random.

The book contains many configuration file examples, man page extracts and perl script listings but these look to have been included largely as padding. They use the monospaced font which contributes to an overall look of poor typography. The way that abbreviations are dealt with is very inconsistent. For example even up to the last chapter the book reminds us that LDAP stands for Lightweight Directory Access Protocol!

There is no troubleshooting section which is very odd for a book designed to help in deployment.

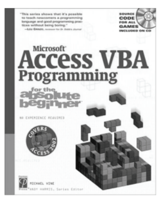
The book has a very dated feel due to several factors; excessive use of man pages, providing details on support for TTY-based email clients such as pine; getting bogged down in details such as how to unpack and build froma compressed tarball.

It is obvious that the author has the necessary knowledge to effectively deploy OpenLDAP including integration with other systems (e.g samba and DNS) but the experience and knowledge of the author is too poorly presented to be of much help to the reader.

Access VBA for the Absolute Beginner

by Michael Vine, ISBN 1-59200-039-8, 328pp + CD, Prima Tech

reviewed by Richard Knight



This book is aimed at absolute beginners and those who have some experience of Access or indeed experienced programmers (who have not yet tried VBA) but is obviously for complete novices. The content is simple and seems to be a bare statement of facts, to the point where it almost appears like a set of classroom notes.

The book contains a CD-ROM containing all the code examples. However, in the latter half of the book several pages list the full code again, where he leaves the reader to read through and understand. The assumption being that he explained it beforehand therefore the beginner requires no more help. Every chapter sets some tasks, however the “answers” are missing. No explanation is given how to run code other than through On Click events, yet he devotes a chapter to debugging and explaining how to step though code.

Given that this is a book about using Access and VBA, it is disappointing that we are well over halfway through the book before he introduces data files. He introduces SQL yet fails to then explain how the user integrates it into VBA.

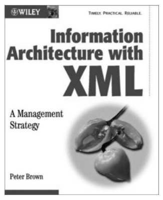
The book contains a CD-ROM containing all the code examples and a short Links section giving some potentially useful VBA programming sites for the reader to follow up.

This is not a bad book but it appears to be trying to cover too many bases. For a complete novice to Access and VBA it could be useful if backed up with some other help, I hesitate to say “Not Recommended” but do not think it warrants “Recommended” status. For his other two reader categories, I would have to say “Not recommended.”

Information Architecture with XML

by Peter Brown, ISBN 0-471-48679-5, 324pp, Wiley

reviewed by Christopher Hill



Should you choose to build a house, you might read “Build your own house in one weekend”, which would include instruction on using the range of tools that you need. The book would be of little use if every reference to the hammer was expanded to a paragraph or two, and the other tools only get passing references.

When building a corporate Information structure you need to manage Meta data, dictionaries, cross-references, context, versions, formats etc. You will have to choose the tools to use, structure them for ease of use and then motivate staff to use the tools. You will have to consider how knowledge will be retrieved from the data that you have stored in your system.

This book is at management strategy level; there is very little technical content. The above issues are all lightly touched on, but viewed through the XML telescope. It feels like “The answer is XML – now what was the question?”

The author is head of Information Resources Management in the European Parliament, and so speaks from wealth of experience of a very large project. He considers the planning and implementation of Information Management urging wide consultation to build the vocabulary, stressing the importance of naming conventions, datatypes and schemas. He then considers how to move old data into the new format (XML of course!) while maintaining pace. The final flourish covers Web Services, delivery management and navigation strategies.

I am not sure who would want to read this book. If your interest is in Information Architecture then there are many better books on this subject; or if XML again better books are to be found. This book might find use as a general introduction, but it will not become a long-term reference.

The Web

Web Design Tools & Techniques

by Peter Kentie, ISBN 0 201 71712 3, Peachpit Press

reviewed by Christopher Hill

As I was reading this book the saying “Jack of all trades – master of none” kept running through my head.

44 ‘chapters’ in 436 pages – with a chapter on Java programming, another on JavaScript, and another on Shockwave Movies – you begin to get the idea that this book dips a toe into just about every tool and/or technique that the author could find.

This is a second edition, published in 2002, yet the impression I get is of a much older book. The chapters on HTML talk of the problems of Internet Explorer 2, and that nearly all browsers’ work to HTML version 3.0 – HTML 4.1 was released in 1999!

There are examples of very bad style; of using a fixed template so that you can use `<H2>` for document title and `<B>` for sub-headings.

There are many errors; suggesting that `<ROWSPAN>` and `<COLSPAN>` attributes can be used in `<TR>` elements; or if you want larger coloured text you use two `<FONT>` tags and only one closing `</FONT>` tag; much of the HTML is not well formed.

The first tool that the author suggests that budding web designers might use is Microsoft Word 2000, which is generally recognised to produce very bloated HTML.

This is an old book with a little editing and a few more recent tools added to produce the second edition, which tries to address too many topics. Not recommended.

More Eric Meyer on CSS

by Eric A. Meyer, ISBN 1-7357-1425-8, New Riders

reviewed by Francis Glassborow

If you have read the author's previous book *Eric Meyer on CSS* (cascading style sheets) you will already know whether you like his presentational method. If you have and do, you will need no persuading from me to purchase this sequel. Now for everyone else.

If you already know the basics of using CSS and want to progress a little further this might be a book for you. The author uses a purely project based approach. This book consists of ten projects.

The first project in the book consists of taking an existing page, designed and implemented with heavy HTML usage and convert it to one that uses a CSS driven layout. I.e. the web-page equivalent of converting source code written in K&R C into and OO design using C++.

In each project the author walks you through the process. The author is honest in that he points to the things that can go wrong if you slavishly apply the process somewhere else.

I like this style of learning/teaching but not everyone agrees. If it suits the way you learn and you want to spend time improving your web-pages or acquiring skills in this area this would be a good book for you.

Web Caching

by Duane Wessels, ISBN 1 56592 536 X, O'Reilly

reviewed by Christopher Hill

Internet browsing is a wonderful thing; your browser requests a page and ‘the internet’ returns it to you. Lots of computers around the world, co-operating to respond to these requests efficiently enough to be commercially and socially useful.

Much of this efficiency is gained by storing the results of previous requests in caches within your browser, but also in caches in “the internet” between your browser and the server holding the original page being requested.

The bulk of the book covers issues to do with the caches beyond the browser. Why have caches? That is a necessary component of a firewall; company control and filtering; reduce network traffic; to speed up response times. The politics, copyright and legal (from USA point of view) problems are rehearsed.

A considerable part of the book covers the theory of caching - what do you cache in the first place and what do you discard when the cache is full. How do you get people to use your cache – manual configuration of the browser (fine in a company, but hard work); let the browser discover your cache; to hijacking HTTP requests via an interception cache.

Throughout the book the caching request and response headers are explained in considerable detail. The protocols for caching hierarchies are also covered.

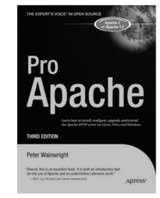
Most of these issues have very specific interest for quite a limited range of people. Of more general interest are the discussions for the web site administrator who either wants to make the most of caching (to give a prompt response), or who wants to ensure their pages are not cached (e.g. to get accurate site use statistics). There is also a useful discussion on how to get the best of both worlds.

The book is well written and logically presented. Recommended if you want to make the most of your high-volume web site, or gain an understanding of the bit of “the internet” between the server and the browser.

Pro Apache 3ed

by Peter Wainwright, ISBN 1-59059-3006, 880pp, Apress

reviewed by Alan Barclay



Pro Apache is the latest revision of Peter Wainwright’s bestselling book on Apache configuration and administration. It is a thoroughly comprehensive and large volume covering a good number of topics including much detail about installation, configuration, maintenance and deployment for both v1.3 and v2.0 of the Apache HTTP Web Server.

The book starts out with interesting background information about HTTP and the operation of Apache and then several pages on the underlying TCP/IP concepts. The latter I feel should probably be left out from what is already a big book. It then continues to provide detail about all of the configuration capabilities along with many useful examples.

Following this are chapters on options for supporting dynamic content (via SSI, CGI, PHP, Tomcat/Java), implementing User Authentication, improving performance and enhancing security. All of which are extremely pertinent issues for an aspiring web master. As a newbie I was successful in getting my Solaris 9 Apache package configured and up-and-running along with PHP in no time.

Peter talks authoritatively about his subject and I am filled with confidence about the advice that he is giving. Unfortunately two small aspects let the book down a little.

Firstly, I found the formatting of the Sections and Sub-Sections titles difficult to differentiate which caused some confusion about where I was, despite the book following a common publishing style. Otherwise it is perfectly well formatted and provides information clearly and concisely Secondly I found a small number of errors in the examples and the table of contents.

I am left thinking that this book was rushed to print without enough care and attention given to the final copy and despite being published in January 2004 I was unable to find an errata or source code download on the Apress website. Overall this is a book with great technical content but some publishing glitches.

ACCU Information

Membership news and committee reports

View From the Chair

Ewan Milne
chair@accu.org

This, I am sad to say, is my last From The Chair. After three years I feel it is time to step down from the post of Chair. I have got a great deal out of the job, and I hope I have given just as much. An increasing work load at my day job, however, has made it difficult to give the role the attention it really deserves (incidentally, since I made the decision, that load has only increased, so I know I'm making the right choice). Also my "secondary" role as Conference Chair has taken up much of the time I have to devote to the ACCU. In fact I find this the most engaging and rewarding task, and so it is my plan to devote my attention to it – while bandwidth allows. You may be reading this just before the AGM, and so while nominations are still open. I will only say that I have made a nomination for the post of Chair, and feel that the person who has accepted this will do an absolutely excellent job.

What I am glad to note in this last column is a flurry of visible activity which is the culmination of much background work by, it must be said, several others: notably members of the committee giving up their spare time. The new website has been very successfully launched, thanks to the efforts of Allan Kelly and our new web editor, Tony Barrett-Powell. Incidentally, I'd like to say that Allan, in his column last issue, was unduly hard on himself. He took hold of a project to develop a new and modern website which had failed to get beyond the talking shop, and has finally delivered the goods. It may have not have been a wholly smooth process, but as a member told me, full time project managers have had much worse happen. Also making a great contribution has been Tony. His role as web editor now has a higher profile, but it should be mentioned that he has been in charge of converting the journals back issues to XML, and doing almost all the conversions himself, which has been quite a major ongoing task. A final thank you to Tim Pushman, who has carried out the development work on the site through his company Gnomedia. While he has of course been paid for this, I am told that he is doing several extra tasks on a voluntary basis, as a member.

Also, you will hopefully notice a whole new look to C Vu this issue. Pete Goodliffe has worked with Alison Peck, our Production Editor, to redesign the magazine. We hope that it provides a whole new fresh look, and we hope that you like it – please let us know your feelings. Just be thankful that the mugshot of your outgoing Chair only has to make a single appearance.

And so it it just remains for me to thank everyone for your input into the Association over the past three years, and ongoing. I'll see you all at future conferences.

Secretary's Report

Alan Bellingham
secretary@accu.org

I rarely miss the committee meetings, but I missed the February one, due to feeling 'under the weather', and being distinctly unwilling to drive in the London area when my wits weren't of the best. So this report is based on the minutes taken by David Hodge.

The meeting this time took place in Weybridge, hosted by your standards officer Lois Goldthwaite.

Once past the usual attendance, apologies, reading of the minutes and discussion of matters arising, the main business started with the reports. Of these, the most interesting points are probably that the Freepost account will be closed once the just-renewed period expires (it's hardly used, and would need moving anyway), that membership is at just over 850, with the usual pre-conference bulge, and that there has been a bit of a brouhaha over the UK C++ Panel's objections to the proposed additions to C++ (Microsoft's recent activities have been a little controversial in some quarters).

As far as the conference is concerned, all appears to on course.

The new website has gone live. There are some teething problems - some items have yet to complete the transfer from the old site - and we have to address accessibility issues, but it should be completed over the next few months.

There was some discussion over the demise of the CUJ magazine, which was wound up recently. Since there are both readers and writers without a home any more, there was some consideration over whether we could, perhaps should, exploit this hole in the market, perhaps by attempting to turn our existing journals into a professional publication.

Finally, the AGM beckons.

The committee has decided to propose Reg Charney as an honourary - perhaps honorary? - life member, in recognition of his work in setting up and running, for so many years, our US group.

Also, as you may be aware, Ewan Milne is stepping down as Chair as of the AGM. He has proposed Jez Higgins to replace him and so, under the new rules, that proposal has been seconded and will be voted upon at the meeting. All other officers will be standing for re-election, and a list thereof, with proposers and seconders, will be issued at the meeting. Under the rules as changed last AGM, such pre-proposals no longer prejudice any proposals from the floor of the AGM, but they should allows the meeting to spend less time in casting around for candidates, and thus allow more time to discuss whatever other matters we may have.

Our next meeting is currently scheduled for the 20th May, a few weeks after the AGM.

Membership Report

David Hodge
membership@accu.org

Since the beginning of the year there have been some 50 new members, these all generated by the conference we are running in April, our membership now stands at 879.

For the benefit of those new members and for those that might have forgotten, please send all changes of mail and email addresses to me.

If you have missing journals then I am also the one to contact.

If you would like to consider being the membership secretary from April 2007 please contact me. We are looking to add a large amount of automation on the new website to handle membership, you could help in its design and testing.

Website Report

Allan Kelly
allan@allankelly.net

Just over two weeks ago we launched the new ACCU website! I expect most of you have already visited the site, and on the whole comments are positive, but just in case you haven't check out the new www.accu.org.

The site was finally launched on 14 February, it is now the start of March and we've had just short of 130,000 page visits. Incredible I know, some of them will be bots but still, that is a lot of visitors.

I have to say a big thank you to those most closely involved: Tony Barrett-Powell the ACCU web-editor and Tim Pushman of Gnomedia who have done most of the work. Jez Higgins has been a great help in the past few months, additional thanks for various supporting work go to Alan Lenton, Ian Bruntlett and Paul Johnson.

We also have a new book database system. This has been developed by Parthenon Computing and is linked to bookshops and carries adverts. Parthenon will be paid from the revenue generated with any extra revenue be split between the ACCU and Parthenon. So, if you are buying a book please buy it through the site.

The new site isn't the end of the story. We still have work to do. The whole point of redeveloping the site and installing a new CMS system was to allow us to keep the site up to date and us it as a new journal media.

Still, there is pressing work to do, we need to move the US ACCU site over, mailing lists, mail archives and journals have yet to be moved from the old server. And there is more.

Again, thanks to all those who have helped. ■