The magazine of the ACCU

www.accu.org

{cvu}

Volume 22 Issue 4 September 2010 £3

Features

Regulars

JOIN THE ACCU!

You've read the magazine, now join the association dedicated to improving your coding skills.

The ACCU is a worldwide non-profit organisation run by programmers for programmers.

With full ACCU membership you get:

- 6 copies of *C Vu* a year
- 6 copies of *Overload* a year
- The ACCU handbook
- Reduced rates at our acclaimed annual developers' conference
- Access to back issues of ACCU periodicals via our web site
- Access to the *mentored developers projects*: a chance for developers at all levels to improve their skills
- Mailing lists ranging from general developer discussion, through programming language use, to job posting information
- The chance to participate: write articles, comment on what you read, ask questions, and learn from your peers.

Basic membership entitles you to the above benefits, but without Overload.

Corporate members receive five copies of each journal, and reduced conference rates for all employees.

How to join

You can join the ACCU using our online registration form. Go to **www.accu.org** and follow the instructions there.

Also available

You can now also purchase exclusive ACCU T-shirts and polo shirts. See the web site for details.

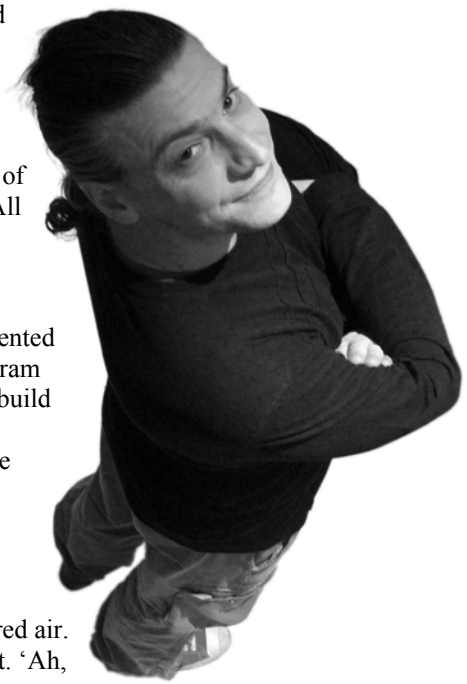
Design: Pete Goodliffe

PERSONAL MEMBERSHIP
CORPORATE MEMBERSHIP
STUDENT MEMBERSHIP

PROFESSIONALISM IN PROGRAMMING
WWW.ACCU.ORG

{cvu}
Volume 22 Issue 4
September 2010
ISSN 1354-3164
www.accu.org

A Simple Assignment



Heavens, Holmes! I sometimes wonder if you need air at all!' I fought through the thick atmosphere of pipe smoke in our living room at 221b Baker Street to open a window. 'How long have you been there?'

He was sitting at the writing desk, with two long rolls of paper side by side, as if he were balancing accounts. 'All night, Watson,' he replied wearily. 'It must be here! I have the code listing and the build log, and yet! I am overlooking something obvious, I must be!'

Clearly he was looking into the little C# problem presented by Lady Harrington. The code compiled, and the program ran, but produced strange output results. The original build log was at level-1 warnings only, which had elicited a withering response from Sherlock Holmes. And so, the level-4 build log was here.

'Almost as long as the code listing, I see,'

'You are scintillating, Watson,' he jibed. 'It is like attempting to locate a wasp in a beehive.' He puffed furiously at the pipe, as if offended by the newly-cleared air. At that moment, Mrs. Hudson chose to bring breakfast. 'Ah, Mrs. Hudson, a well-timed cup of tea!' exclaimed Holmes. 'Perhaps you'd care to look over this code and see what it is hiding, for I am certain that I cannot?'

'Certainly, Mr. Holmes. Just as long as you open another window before we all suffocate!' replied our long-suffering housekeeper with some asperity, setting the tray on the breakfast table. No sooner had Holmes and I made ourselves comfortable than Mrs. Hudson looked up again. 'This, surely, is the problem,' she asserted. 'This warning about 'Assignment in conditional expression'. Yes, I am sure, here it should be 'equals-equals' instead of just 'equals'.'

Holmes recovered quickly and with a wry expression remarked 'Let that be a lesson to us all: never try to out-stare bad code on your own.'

STEVE LOVE
FEATURES EDITOR

The official magazine of ACCU

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.

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 programmers, for programmers – and have been contributed free of charge.

To find out more about 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

SUBMISSION DATES

C Vu 22.5: 1st October 2010
C Vu 22.6: 1st December 2010

Overload 100: 1st November 2010
Overload 101: 1st January 2011

WRITE FOR C VU

Both C Vu and Overload rely on articles submitted by you, the readers. We need articles at all levels of software development experience. What are you working on right now? Let us know!

Send articles to cvu@accu.org. The friendly magazine production team is on hand if you need help or have any queries.

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

People Power

Pete Goodliffe shows us how to become better programmers.

Two things are infinite: the universe and human stupidity; and I'm not sure about the the universe.

– Albert Einstein

This is the 64th 'Professionalism in Programming' column I've written for C Vu. If you are observant you'll have noticed a subtle but significant change that I have introduced after 63 issues. Have you spotted it yet?

Sidle over to the sidebar for the answer. And if you're not at all curious, then skip it and read on...

Programming is a People Pursuit

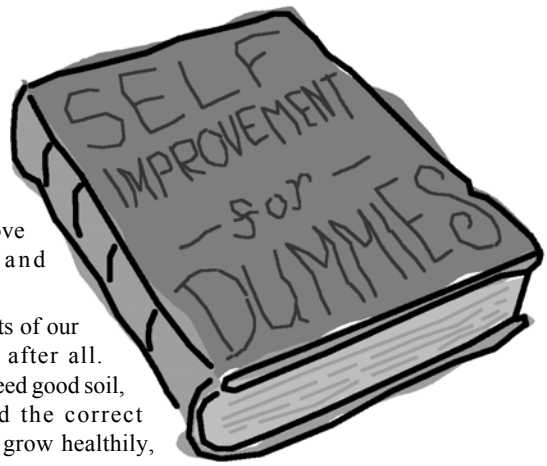
Almost since the first programs were constructed we have realised that programming is not a solely technical challenge. It is also a social challenge. Software development is a pastime that involves writing code *with* other people, *for* other people to understand, working with other people's (variable quality) code, joining or leaving software teams, managing developers (which is rather like herding cats), and so on.

Many of the most enduring programming tomes are devoted to the people problem, for example: *The Mythical Man Month* [1], and *Peopleware* [2].

So, based on this, the first simple technique we'll consider that will help us to becoming better programmers is this:

> Purposefully place yourself beside excellent programmers.

That is, if you want to be an exceptional programmer (or remain one) then you must consciously place yourself daily amongst people who are exceptional programmers. It's a really simple but profound way to make sure that you improve your skills and attitudes.



We are products of our environment, after all. Just as plants need good soil, fertiliser, and the correct atmosphere to grow healthily, so do we.

Spending too long with depressing people will make you feel depressed. Spending too long with run-down people will make you feel tired and lethargic. Spending too long with sloppy workmen will encourage you to work sloppily yourself – why bother trying if no one else is? Conversely, working with passionate individuals who strive to make better software will encourage you to do the same.

By immersing yourself in the environment of excellent programmers you will treat yourself to:

- Enthusiasm that is infectious
- Motivation that is inspirational
- Responsibility that is contagious

Find people like that and marinate yourself. Consciously seek out the people who care about good code, and about writing it well. In that kind of environment, you won't fail to be nurtured and encouraged.

By working with high calibre developers you will gain far more than technical knowledge, although that in itself is very valuable. You'll enjoy positive reinforcement of good programming habits and attitudes. You'll be encouraged to grow, and challenged to improve in areas you are lacking. This isn't always comfortable or easy, but it is worthwhile.

So make a point of seeking out the best programers and work with them. Design code with them. Pair program with them. Socialise with them.

What to do

You could make this kind of relationship formal with *mentorship* (many workplaces try to put mentorship schemes into practice formally). Or you may pursue it informally: get yourself assigned on the same projects as the great programmers. Move company to work with them. Go to conferences, talks, or user groups to meet with them. Or just make a point of hanging out with other great programmers.

As you do this, try to learn from them:

- How they think and solve puzzles
- How they plan a route into problems

As good as a rest

So have you noticed what's changed? Look at the top of the page. I've changed the series title.

Why have I done this?

I started the 'Professionalism in Programming' column ten years ago specifically because that was the strap line of the ACCU. Our maxim. There's an awful lot that comes under the umbrella of *professionalism*, and I've made reasonable headway in examining some of it. Remember that this professionalism has never been about merely being paid to write code. No, 'professionalism' here means something far deeper. It's about taking a responsible, mature attitude to the design, construction, cultivation, and crafting of software systems. These days the nearest fashionable term is software craftsmanship. Although that term is not without its own titular controversy.

Since I just had to describe it above, it's clear that 'Professionalism in Programming' is an ambiguous series title. It's not immediately obvious exactly what it means. All along, I've been mining specific tips, techniques, issues and practices that will help us (myself included) to become better programmers.

And so, with that in mind, I have chosen to switch the series title over to the far clearer and more direct 'Becoming a Better Programmer'. Hopefully you'd not become too attached to the old title. I hope you like this new one, and will enjoy what I write under the new banner.

This is not only a change of the words in the magazine header, though. I also intend to change the format of the material I write. In order to focus on specific things that will help us to become better programmers, I will write each time about a single, simple key that will help us to improve as programmers. This will be something small and practical that will be easy to apply to our daily coding regimen.

Each issue I will also attempt balance this with a *caution*. This is a technique, attitude, or problem that will prevent or hinder you from becoming a better programmer.

Hopefully this is a simple recipe that will work well. But nothing is ever set in stone.

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@goodliffe.net

- The attitude they adopt when things get hard
- How they know to keep pushing on a problem, when to take a break, or when try a different approach
- Specific coding skills and techniques you don't understand

An interesting and beneficial side-effect of working with good coders is that you are far more likely to end up working with good code.

Know your experts

Consider carefully what you think an excellent programmer looks like.

It might be the *expert* level practitioner described by the *Dreyfus Model of Skills Acquisition* we saw in [3]. (Beware, they might be quite hard to work with, or hard to learn from!) Or it could perhaps be a relative novice who doesn't 'know it all' but has innate skill and an excellent attitude.

You specifically *don't* want to get alongside people who work too hard, filling all the hours God sends with code. Those people are almost certainly not the exceptional programmers! Managers often think that employees who spend every waking hour on the project are the programming heros, but often this really shows their lack of ability. They can't get things right first time, so they have to spend many more hours getting the code to 'work' than was actually necessary.

Experts make it look easy and get things done on time.

20/20 hindsight

I look back over my career and realise that the most enjoyable and personally productive times I've encountered have been when I've been working alongside excellent, motivated, interesting developers. And because of this, I will now always attempt to place myself alongside people like that.

I've realised that they make me better at what I do, and I have more fun whilst I'm doing it.

So ask yourself:

- Are you around people you think are excellent developers right now? Why? Why not?
- How can you move yourself nearer better coders? Can you move projects or teams? Is it time to move company?
- Perhaps you're not yet employed. In education are you being taught by people who know their onions? Are you working on the best open source projects?

Caution: Don't stagnate

Iron rusts from disuse; water loses its purity from stagnation... even so does inaction sap the vigor of the mind.

– Leonardo da Vinci

The small caution that I will leave you with in this column is this:

> Be wary of stagnation. Seeking to become a better programmer, by definition, is not the most comfortable lifestyle.

Few people make conscious decisions like the one above to move themselves into the path of excellent programmers. It's risky and hard. Keeping in the same job is usually easier, more familiar, and convenient. And in the current economic climate, it's also the safer bet.

However, there is a danger in staying in one place too long, doing the same thing over and over with no new challenges. All too easily, we get entrenched in what we're doing. Familiarity breeds comfort. We like being local experts; the king of our little coding castle.

When you are unnecessarily comfortable you have no need to step out of that comfort zone. No need to learn new things, no need to be stretched, or to even acknowledge that there is more to learn.

Beware of this trap.

Even if you're not considering moving jobs, work out how to prevent stagnation. Can you work on new projects? Get into a part of the system you've no experience in? Can you attempt a new task that requires a different skill?

Homework

1. Act on these points:
   - Are you alongside excellence? Can you move?
   - Are you stagnating? How can you shake this up?
2. Look back over your programming career. What is the single most important thing that you think has made you a better programmer? If there is something specific that you can put your finger on, I would love to hear about it. Please send me an email. ■

References

[1] *The Mythical Man Month* Frederick P. Brooks Jr. Addison Wesley, 1995. ISBN: 0201835959.

[2] *Peopleware* Tom Demarco and Timothy Lister. Dorset House Publishing, 1999. ISBN: 0932633439.

[3] 'Professionalism in Programming #60: Live to Love to Learn, Part 2' Pete Goodliffe. In: *C Vu* 22.1, March 2010.

Flexible Function Façades for C and C++

Matt Wilson presents a technique for compiling headers as C or C++ without changes.

This article illustrates a number of simple techniques for writing (header-only) library function façades that must compile and function correctly as both C and C++. It covers issues of ordering and comparison, conversion to and from string forms, working with multiple character encodings, and pointer/reference equivalence, focusing on intrinsic software quality characteristics of correctness/robustness and modularity. It also examines areas in which use of such façade function suites from C++ can be substantially enhanced by the application of namespaces, overloading, default arguments and function templates, raising the level of abstraction and increasing flexibility.

Introduction

> Quoth C++ programmer to C programmer: Your language is primitive, inexpressive, dangerous, and engenders resource loss.
>
> Quoth C programmer to C++ programmer: Your language is baroque, complex, flabby, obtuse, and produces inefficient and bloated executables.

All too often, proselytisers of a given language ignore the hazy margins where it may need to interact with others. Heaven forfend that programmers might actually need to work in anything other than the 'one true language'!

In large part, the business of software development is the business of writing abstractions to solve your current problem, and then packaging them for reuse so you don't have to reinvent the same solutions in the future. In other words: writing libraries. This is an essential task, but not always an easy one. It's complicated considerably when you need to write a library that will serve more than one language.

If you happen to be writing in a .NET language, then you have less to worry about (in this regard, anyway): just pump out your assembly and (with certain limitations that are outside the scope of this article) consume it in C#, F#, C++/CLI. (I don't include VB.NET because it is a bad joke of a language built from, yet mocking, an execrable 'language' whose very existence speaks to much of what is wrong with the software development world. So there.)

A similar situation exists with Java and Scala (and no doubt other JVM languages of which I have neither awareness nor experience). There are some less obvious, but no less useful, combinations, such as using COM as an interoperability mechanism between C/C++ and .NET, Python and Ruby. (The latter combination is one I use all the time in internal tools. Who said COM was dead?!)

But this is the ACCU, where the Cs (used to) stand for C and C++, so I expect the majority of you are thinking: What about C++ being a (nearly proper) superset of C? Isn't that the very essence of inter-language code reuse? Well, yes. But, then again, no.

As well as being *the portable assembler*, C is also considered to be the lingua-franca of interoperability. C APIs can be consumed directly from C++, D, Delphi, Fortran, and others. (It can be consumed indirectly from C++/CLI and C#, whereby the compilers and the .NET runtime do all the things for you that you have to do yourself to consume it from the likes of Java, Perl, Python, Ruby.)

One problem with direct (and indirect, for that matter) consumption of C is that the level of abstraction of a C-API is usually lower than that convenient to the application programmers of the consuming language. You need look no further than many standard library functions. For example, say you want to erase a file whose path you're storing in a string object:

```
std::string path = "~/stuff";
::remove(path.c_str());
```

The C++ code is written with `std::string` as the unit of currency for text strings. The C standard function `remove()` requires a C-style string, necessitating our explicit call of the `c_str()` method. This is an example of a phenomenon I call *abstraction dissonance* [1], and I believe it seriously detracts from expressiveness and flexibility.

There are several ways in which C–C++ cross-language library support can be more clearly provided. In my experience, the most common and generally the most useful is to write adaptation layers. For our trivial `remove()` example this could be achieved as follows:

```
inline int remove(
  std::string const& path
)
{
  return remove(path.c_str());
}
```

If it didn't violate the C++ standard rules, we might wish to put this in the `std` namespace.

However, in some circumstances a better approach is available. Bearing in mind that C programmers can experience just as much vexation with verbose low-level C-APIs, sometimes it's useful to write façades for C-APIs in C.

The problem

For pedagogical purposes we'll consider a more technology-specific API. At the moment I'm doing expert witness work, and am having to conduct large-scale comparative source code analyses partially aided, thankfully, by automated code analysis tools. One of Synesis' extant source analysis tools is having a few superchargers fitted, in order to cope with the enormous data sets involved in a case on which I'm working. One of the modifications required is to represent paths as fixed-length unique identifiers. Since the dedicated machine on which I'm executing the analyses is a Windows box, a GUID fits the bill nicely. In order to work with the various data structures involved, I need to provide the following operations for GUIDs:

- Conversion to text string
- Conversion from text string
- Equality comparison
- Ordering comparison

Conversion to string is available from the Windows API functions `StringFromCLSID()` and `StringFromGUID2()`, defined as follows.

MATTHEW WILSON

Matthew is a software development consultant and trainer for Synesis Software who helps clients to build high-performance software that does not break, and an author of articles and books that attempt to do the same. He can be contacted at matthew@synesis.com.au.

```
WINOLEAPI StringFromCLSID(
  REFGUID   rclsid // GUID (aka CLSID) to be
                   // converted
, LPOLESTR* ppsz   // Address of string pointer
);
int StringFromGUID2(
  REFGUID  rguid   // GUID to be converted
, LPOLESTR lpsz    // Pointer to buffer
, int      cchMax  // Size of buffer
);
```

Both work only with wide strings, and are therefore inconvenient if you're working with multibyte strings. **`StringFromCLSID()`** is doubly irksome because it allocates the result string – even though it must always have a length of 38 – which you must then free. Since I am working with multibyte strings I don't want to have to write code such as the following, from which I've elided error handling for brevity:

```
/* (i) with StringFromCLSID() */
GUID      guid = . . .
char      mbs[39];
LPOLESTR  olestr;
HRESULT   hr = StringFromCLSID(guid, &olestr);
size_t    n  = wcstombs(mbs, NUM_ELEMENTS(mbs),
   olestr);
CoTaskMemFree(olestr);
fputs(mbs, stdout);

/* (ii) with StringFromGUID2() */
GUID      guid = . . .
LPOLESTR  olestr[39];
char      mbs[39];
HRESULT   hr = StringFromGUID2(guid, olestr,
   NUM_ELEMENTS(olestr));
size_t    n  = wcstombs(mbs, NUM_ELEMENTS(mbs),
   olestr);
fputs(mbs, stdout);
```

Yuck. And imagine how much worse it is if the error-handling is in there. About as transparent and expressive as a teenager!

If you're working in C++ this issue is already resolved by use of STLSoft's *string access shims* [2] [3] [4]: the **`GUID`** overloads are available via comstl/shims/access/string/guid.hpp.

```
#include <comstl/shims/access/string/guid.hpp>

GUID guid = . . .

::puts(stlsoft::c_str_ptr(guid) , stdout);
::fputws(stlsoft::c_str_ptr(guid) , stdout);
```

That may not look so impressive when used explicitly, but if you're using FastFormat (for output) or Pantheios (for diagnostic logging) then the advantage is much more significant, since they both understand anything for which string access shims are defined (and do so with 100% type-safety [5]):

```
GUID guid = . . .

ff::fmt(std::cout, "guid = {0}", guid);
pan::log_DEBUG("guid = ", guid);
```

It's also possible to use a façade class, such as STLSoft's **`comstl::guid`** class:

```
#include <comstl/util/guid.hpp>

GUID guid = . . .

::puts(comstl::guid(guid).c_str(), stdout);
::fputws(comstl::guid(guid).c_str(), stdout);
```

REFGUID

As anyone who's read *Imperfect C++* [2] will attest, I've been intrigued by the nitty gritty of how C and C++ interact with each other for a long time. One of the things I learned early on is the relationship between pointers and references. Though not prescribed by the standard, all known compilers treat the two as equivalent in object code. In other words, a C++ reference to a given type is represented as a machine word containing the address of the object it is referencing, just as a pointer is. This can be used to advantage.

In COM, a GUID is a 128-bit type, comprised of one 32-bit unsigned integer, two 16-bit unsigned integers, and eight 8-bit unsigned integers. In almost all cases where a **`GUID`** is passed to a function, a **`NULL`** pointer is neither appropriate or acceptable. Because COM is a multi-language binary standard, its API functions can only represent the minimum common set of linguistic constructs. In simplistic terms, COM has a C-API.

In order to simplify C++ client code's manipulation of **`GUID`** constants, while still observing the requirement to be a pointer in C, many COM functions utilise the fungible type **`REFGUID`** (and its aliases **`REFIID`** and **`REFCLSID`**), which is defined as **`GUID const*`** in C and **`GUID const&`** in C++, relying on the binary equivalence of pointers and references provided by the Microsoft and other COM-compatible compilers.

In other words, the definition of the aforementioned API function **`StringFromCLSID()`** is seen in C compilation units as

```
WINOLEAPI StringFromCLSID(
  GUID const* rclsid
, LPOLESTR * ppsz
);
```

and in C++ compilation units as

```
WINOLEAPI StringFromCLSID(
  GUID const& rclsid
, LPOLESTR * ppsz
);
```

This saves on a whole lot of otherwise-necessary address-of operators applied everywhere a **`GUID`** (aka **`IID`**, aka **`CLSID`**) is specified. Although only one character – **`&`** – this actually does result in significantly clearer code, and I think the COM team made the right (albeit implementation-defined) choice.

This requires two temporary instances be created, which smells bad to me, even though actually it isn't bad in this case. Furthermore, it involves my arch-nemesis, **`c_str()`**, in application code, which is invariably an indicator of abstraction dissonance.

Either of these two types of solutions are viable in C++, but are of no use to C application programmers.

In a similar vein, **conversion from string** is available from the Windows API function **`CLSIDFromString()`**:

```
HRESULT CLSIDFromString(
  LPOLESTR lpsz   //Pointer to the string
                  //representation of the CLSID
, LPCLSID pclsid  //Pointer to the CLSID
);
```

There's no issue of memory ownership here, but still the inconvenience of working with multibyte strings. (I won't bother with an example here, as I'm sure you can all, gentle readers, imagine the tiresome application of **`mbstowcs()`** in application code.)

Neither **equality comparison** nor **ordering comparison** are provided in any specific form for either language. They can both by synthesised by use of **`memcmp()`**, but, once again, we're leaking abstraction levels into application code (i.e. by highlighting that a GUID is a baroque melange of integral types with **`memcmp()`** and **`sizeof`**; see sidebar).

So, I had a clear requirement for GUID to be 'promoted' to a *value type* [2], for comparison at least, and to be succinctly and flexibly converted to and from text string representations. Furthermore, since STLSoft provides façades in C form wherever practicable, I elected to beef up the GUID handling components with a suite of functions supporting both C and C++.

Listing 1

```
/* copyright information header block */
/* includes */
/* namespace (opening) */
/* C functions (visible from C) */
/* C functions (visible from C++) */
/* C++ functions */
/* namespace (closing) */
```

This article is the story of the design decisions involved. If you're using COM, you may find the technical details of the following discussion directly relevant. If so, that's a bonus, but not my expectation for most readers. Rather, the purpose of this article is to discuss several principles and techniques for writing façade components that support C and C++, applicable to any application domain.

Flexible function façades for C and C++

Let's now look at the implementation of the new/enhanced GUID function façades available with **STLSoft 1.10**. Given the limited space in this forum, I'm going to cover the broad structure of the file, and then break it down into more detail to cover the individual functions.

The function façades are defined within the file `comstl/util/GUID_functions.h`, which has the structure shown in Listing 1.

Includes

As indicated by the comments, this file supports both C and C++ compilation units, so you can expect **`__cplusplus`** to put in an appearance, at least. The includes include (ha ha!) the first interesting point, shown in Listing 2.

Put aside the STLSoft 1.10 alpha includes, which will change when STLSoft 1.10 enters beta, the 'regular' STLSoft includes shows inclusion of the **COMSTL** sub-project main header, in order to get hold of the COM standard headers (`objbase.h`, `oleauto.h`, `oaidl.h`, ...) and various constants and macros. It also, conditionally, includes the default STLSoft header for string access shims [2] [3] [4]. For those of you familiar with my use of string access shims, this is a clear indication that we'll be promoting string-related flexibility once we hit C++.

Namespace

The multi-language support aspect is equally evident in the namespace blocks, as shown in Listing 3.

In C++, we play nice with the rest of the world by encapsulating everything within the **`comstl`** namespace; in C, we rely on the naming convention; see the next section. (There's actually a little more to it, but it's not relevant here; see [6], and look inside the headers, for more details).

C function declarations

Now we get to the meat of the C functions. I'll first show them as a collective in summary in Listing 4.

Listing 2

```
/* STLSoft 1.10 alpha includes */
#include <stlsoft/stlsoft_1_10.h>
#include <stlsoft/quality/contract.h>
#include <stlsoft/quality/cover.h>

/* STLSoft includes */
#include <comstl/comstl.h>
#ifdef __cplusplus
# include <stlsoft/shims/access/string.hpp>
#endif /* __cplusplus */

/* Standard includes */
#include <wchar.h>
```

Listing 3

```
#ifdef __cplusplus
namespace comstl
{
#endif

/* C functions (visible from C) */
/* C functions (visible from C++) */
/* C++ functions */

#ifdef __cplusplus
} /* namespace comstl */
#endif
```

Focusing just on the signatures, we can see two common distinctive features: the **`STLSOFT_INLINE`** storage class specifier; the **`comstl_C_`** prefix.

Be aware that STLSoft, and all its sub-projects (COMSTL, **UNIXSTL**, **WinSTL**, …) are 100% header-only, which significantly simplifies use by avoiding issues of compilation of source and/or distributing binaries. An essential element to achieve that is the use of **`inline`**: as well as being a hint to the compiler to define a given function inline, it also requires that the compiler/linker ensure that only a single version of the function exists, thereby avoiding fatal linker errors. It's the latter aspect for which it's employed throughout header-only libraries.

As we all know, **`inline`** is (and always has been) supported by C++, but was standardised in C only as of C99. Thus, **`STLSOFT_INLINE`** resolves to **`inline`** in C++ and in C-99, or to compiler-specific extensions such

Listing 4

```
STLSOFT_INLINE int comstl_C_GUID_compare(
  GUID const* lhs
, GUID const* rhs
, HRESULT*    comparisonSucceeded
)
{ . . . }
STLSOFT_INLINE int comstl_C_GUID_binary_compare(
  GUID const* lhs
, GUID const* rhs
)
{ . . . }
STLSOFT_INLINE int comstl_C_GUID_equal(
  GUID const* lhs
, GUID const* rhs
)
{ . . . }
STLSOFT_INLINE HRESULT
comstl_C_GUID_from_string_w(
  wchar_t const*  str
, GUID*           pguid
)
{ . . . }
STLSOFT_INLINE HRESULT
comstl_C_GUID_from_string_a(
  char const*     str
, GUID*           pguid
)
{ . . . }
STLSOFT_INLINE HRESULT comstl_C_GUID_to_string_w(
  REFGUID   guid
, wchar_t (*buff)[1 + COMSTL_CCH_GUID]
)
{ . . . }
STLSOFT_INLINE HRESULT comstl_C_GUID_to_string_a(
  REFGUID   guid
, char    (*buff)[1 + COMSTL_CCH_GUID]
)
{ . . . }
```

Afference and efference (aka fan-in and fan-out)

Afference (afferent coupling) refers to the number of other components that a given component utilises, i.e. is coupled to. Efference (efferent coupling) is the inverse, that is the number of other components that utilise a given component. Each is used as a measure of software quality, in ways that are outside the scope of this article. What we can say, however, is that other things being equal, keeping efference of wrapped API functions is good. Writing façades is, in part, about covering up the ugly/awkward/sub-par. Consequently once you've done that once you can save yourself from further effort by reusing your own good works.

(Note: it's not always a good idea to implement in terms of oneself. In particular, when implementing classes that segregate public and internal methods for the purposes of multithreading, contract enforcement, and so on, it can be a fragile and dangerous pursuit. But we're away from the main topic now, and on to issues I'll be examining in detail in forthcoming instalments of the 'Quality Matters' column over in *Overload*.)

as **__inline** and **__inline__**, or, at worst, to **static** for those very old C compilers for which none of the above are supported. (The **static**-fallback case, needed only for very old compilers, works, but has the obvious cost of code-bloat, since multiple instances of the functions are present. They don't clash the linker because they have internal linkage, by dint of the **static** keyword.)

Use of the **comstl_C_** prefix is simply there to provide API disambiguation in the tried-and-trusted C way of prefixing library API functions with some aspect of the library name [7]. Obviously, such things do not guarantee uniqueness, but in practice it almost always works. In previous versions I've transgressed good practice by using prefixes such as **comstl__**, names that, by having two consecutive underscores, are reserved. These are gradually being replaced by standards-conformant ones, as illustrated here.

Before we look at the implementations, I want to talk about the semantics implied by the function signatures:

comstl_C_GUID_binary_compare() does exactly what you'd expect, i.e. behaves like **memcmp()**, returning **<0**, **0** and **>0** to indicate ordering. In fact, it uses **memcmp()**, so is fast, but its result is architecture-dependent: on different byte-ordering architectures the comparison results of two given **GUID**s may differ.

comstl_C_GUID_compare() performs an architecture-independent comparison, by converting the **GUID**s to string form for comparison. And it has the ability to 'fail', as indicated by the **comparisonSucceeded** out-parameter. We'll see how and why shortly.

Both of these functions, and **comstl_C_GUID_equal()**, take arguments of (non-mutating) pointer to **GUID**, rather than **REFGUID** (see sidebar), implying that **NULL** values are valid. This is a weakening of preconditions, which leads to a modest, though idiomatic, increase in complexity of implementation, as we'll see.

The remaining four functions are for conversion between **GUID** and text string forms, in matching pairs for wide string and multibyte string representations. The 'from string' pair declarations are pretty boilerplate: the return type indicates error in format of the string form to be converted. By contrast, the 'to string' pair parameters warrant some discussion. First, the **GUID** is passed as a **REFGUID** (see sidebar), indicating that **NULL** arguments are not valid (and not accidentally possible in C++). Stranger, perhaps, is the destination character buffer parameter, which is a pointer to an array of dimension **1 + COMSTL_CCH_GUID (== 1 + 38)**. This curious construction is quite different from an array or a pointer, to which an array will naturally decay. In fact, it's the very ability of an array to decay into a pointer (see [2] and [8] for more information on this subject) that motivates the use of an array pointer.

Consider a function with the following declaration instead:

```
STLSOFT_INLINE HRESULT fn(
  REFGUID guid
, char*   buff
);
```

The compiler would allow all of the following:

```
GUID guid = . . .
char ch;
char ar1[1];
char ar2[39];
char ar3[100] = "guid=";
/* 1 */ fn(guid, &ch);
/* 2 */ fn(guid, &ar1[0]);
/* 3 */ fn(guid, ar1);
/* 4 */ fn(guid, &ar2[0]);
/* 5 */ fn(guid, ar2);
/* 6 */ fn(guid, &ar3[0] + 5);
```

Of these, only 4, 5 and 6 are well-defined. 1–3 will involved undefined behaviour, and will likely (and hopefully) crash.

You might think that specifying the array size will help:

```
STLSOFT_INLINE HRESULT fn(
  REFGUID guid
, char    buff[1 + COMSTL_CCH_GUID]
);
```

Alas, this is not so. In accordance with the rules of the language(s), the compiler ignores the number and interprets it as being identical to **fn(REFGUID, char[])**, which it interprets as being identical to **fn(REFGUID, char*)**.

In cases where the size is known at compile time it is possible to have the compiler enforce the array size by defining the parameter as a *pointer to an array*. Having the buff parameter be of type **char (*)[39]** means the only parameter it will accept will be the address of an array of dimension 39.

```
STLSOFT_INLINE HRESULT fn(
  REFGUID guid
, char*   buff
);
. . .
char ar4[39];
/* 7 */ fn(guid, &ar4);
```

In my opinion, this is a great boon to type-safety. The three incorrect cases, 1–3, are rejected. Cases 4 and 5 are replaced by case 7. The only loss in flexibility incurred by the substantial increase in correctness/robustness [1] is the inability to write into a part of a (larger) array. I suggest that this is a bargain well made.

C function implementations

Let's now look at the implementations. This is where straddling two languages results in a bit of mess. For the sake of brevity, I will make only cursory mention of things that are idiomatic, or that have been covered elsewhere.

First, **comstl_C_GUID_compare()** (Listing 5).

The features of note are:

- Use of Variable form [9] of Null Object pattern [10] to simplify use of ***comparisonSucceeded**
- Including **NULL** in ordering (where **NULL** is 'less' than any non-**NULL** string)
- Use of pointer-to-array, in invoking **comstl_C_GUID_to_string_w()**, discussed later

Given the foregoing, all that are left to explain are the two macros, **COMSTL_PTR_2_REF()** and **STLSOFT_NS_GLOBAL()**. As you might guess from the name, the former turns a pointer into a 'reference', by applying a ***** in C++ to make a C++ reference, and leaving it as a pointer in C. The latter applies the global scope operator **::** to the macro parameter in C++, and leaves it unadorned in C.

By dint of these two, the function body will correctly see and invoke **wcscmp()** and **comstl_C_GUID_to_string_w()** whether compiled as C or as C++.

Listing 5

```
STLSOFT_INLINE int comstl_C_GUID_compare(
  GUID const* lhs
, GUID const* rhs
, HRESULT* comparisonSucceeded
)
{
 HRESULT comparisonSucceeded_;
 if(NULL == comparisonSucceeded)
 {
  comparisonSucceeded = &comparisonSucceeded_;
 }
 *comparisonSucceeded = S_OK;
 if(NULL == lhs)
 {
  return (NULL == rhs) ? 0 : -1;
 }
 else
 {
  if(NULL == rhs)
  {
   return +1;
  }
  else
  {
   OLECHAR s1[1 + COMSTL_CCH_GUID];
   if(FAILED(*comparisonSucceeded
      = comstl_C_GUID_to_string_w(
      COMSTL_PTR_2_REF(lhs), &s1)))
   {
    return -1;
   }
   else
   {
    OLECHAR s2[1 + COMSTL_CCH_GUID];
    if(FAILED(*comparisonSucceeded =
       comstl_C_GUID_to_string_w(
       COMSTL_PTR_2_REF(rhs), &s2)))
    {
     return -1;
    }
    else
    {
     return STLSOFT_NS_GLOBAL(wcscmp)(s1, s2);
    }
   }
  }
 }
}
```

Given what we've just learned, the implementation of **comstl_C_GUID_binary_compare()** is now straightforward (as shown in Listing 6).

comstl_C_GUID_equal() (Listing 7) is also pretty obvious, with the detail that the 'assumption' contract enforcement requires that the two compare functions agree when it comes to equality (which they may not be able to do with ordering).

Let's now turn our attention to the string functions.

comstl_C_GUID_from_string_w() (Listing 8) is another straightforward façade function, comprising nothing more than parameter precondition contract enforcements, and forwarding to **CLSIDFromString()**. It applies **const_cast** to improve **const**-correctness for users of the façade, to overcome an oversight in the COM API, which is not **const**-correct in many places. Still, otherwise it does nothing much at all, and you may wonder at the point of it: Why not just have users invoke **CLSIDFromString()** (with cast if necessary)? There are four reasons:

- Const-correctness

Listing 6

```
STLSOFT_INLINE int comstl_C_GUID_binary_compare(
  GUID const* lhs
, GUID const* rhs
)
{
 if(NULL == lhs)
 {
  return (NULL == rhs) ? 0 : -1;
 }
 else
 {
  if(NULL == rhs)
  {
   return +1;
  }
  else
  {
   return STLSOFT_NS_GLOBAL(memcmp)(lhs, rhs,
      sizeof(GUID));
  }
 }
}
```

- Consistency (with **comstl_C_GUID_from_string_a()**)
- Reduction of the efferent use of the COM API function **CLSIDFromString()** to 1 (see sidebar)
- Overloading (as we'll see later in this article)

The aforementioned issue of efference is in play in the implementation of its complement function, **comstl_C_GUID_from_string_a()** (see Listing 9). This function takes a multibyte string, which it converts using the Windows API function **MultiByteToWideChar()**, before nul-terminating the result and passing to **comstl_C_GUID_from_string_w()**, rather than calling **CLSIDFromString()** directly (with a second **const_cast**).

Listing 7

```
STLSOFT_INLINE int comstl_C_GUID_equal(
  GUID const* lhs
, GUID const* rhs
)
{
 STLSOFT_CONTRACT_ENFORCE_ASSUMPTION(
  (0 == comstl_C_GUID_binary_compare(lhs, rhs))
   ==
  (0 == comstl_C_GUID_compare(lhs, rhs, NULL)));

 return 0 == comstl_C_GUID_binary_compare(
    lhs, rhs);
}
```

Listing 8

```
STLSOFT_INLINE HRESULT
   comstl_C_GUID_from_string_w(
  wchar_t const* str
, GUID*          pguid
)
{
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
    NULL != str,
    "string parameter may not be null");
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
    NULL != pguid,
    "return value out parameter may not be
    null");
 return STLSOFT_NS_GLOBAL(CLSIDFromString(
    stlsoft_const_cast(LPOLESTR, str), pguid));
}
```

Listing 9

```
STLSOFT_INLINE HRESULT
   comstl_C_GUID_from_string_a(
  char const* str
, GUID*       pguid
)
{
  OLECHAR  ws[1 + COMSTL_CCH_GUID];
  HRESULT  hr = S_OK;
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
   NULL != str, "string parameter may not be
   null");
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
   NULL != pguid, "return value out parameter may
   not be null");
  switch(STLSOFT_NS_GLOBAL(MultiByteToWideChar)(
     0, 0, str, -1, &ws[0], 1 + COMSTL_CCH_GUID))
  {
    case 1 + COMSTL_CCH_GUID:
      ws[COMSTL_CCH_GUID] = L'\0';
      hr = comstl_C_GUID_from_string_w(
         ws, pguid);
      break;
    default:
      if(S_OK == (hr = HRESULT_FROM_WIN32(
         STLSOFT_NS_GLOBAL(GetLastError)())))
      {
        hr = E_INVALIDARG;
      }
      break;
    }
    return hr;
  }
```

`comstl_C_GUID_to_string_w()` (see Listing 10) is pretty straightforward. Points of interest are:

- Dereference of the pointer-to-array to obtain the array to pass to the underlying API function

Listing 10

```
STLSOFT_INLINE HRESULT comstl_C_GUID_to_string_w(
  REFGUID  guid
, wchar_t (*buff)[1 + COMSTL_CCH_GUID]
)
{
  int r;
#ifndef __cplusplus
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
   NULL != guid, "GUID parameter may not be
   null");
#endif /* !__cplusplus */
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
   NULL != buff, "array parameter may not be
   null");
  r = STLSOFT_NS_GLOBAL(StringFromGUID2)(guid,
     *buff, 1 + COMSTL_CCH_GUID);
  if(0 == r)
  {
    return E_INVALIDARG;
  }
  else
  {
    STLSOFT_CONTRACT_ENFORCE_POSTCONDITION_PARAMS
       _EXTERNAL(L'\0' == (
       *buff)[COMSTL_CCH_GUID],
       "buffer must be nul-terminated");
    return S_OK;
  }
}
```

Listing 11

```
STLSOFT_INLINE HRESULT comstl_C_GUID_to_string_a(
  REFGUID  guid
, char (*buff)[1 + COMSTL_CCH_GUID]
)
{
 OLECHAR  wbuff[1 + COMSTL_CCH_GUID];
 HRESULT  hr;
#ifndef __cplusplus
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
   NULL != guid, "GUID parameter may not be
   null");
#endif /* !__cplusplus */
STLSOFT_CONTRACT_ENFORCE_PRECONDITION_PARAMS_API(
   NULL != buff, "array parameter may not be
   null");
  hr = comstl_C_GUID_to_string_w(guid, &wbuff);
  if(FAILED(hr))
  {
    (*buff)[0] = '\0';
  }
  else
  {
    switch(
      STLSOFT_NS_GLOBAL(WideCharToMultiByte)(
      0, 0, wbuff, 1 + COMSTL_CCH_GUID, *buff,
      1 + COMSTL_CCH_GUID, NULL, NULL))
    {
      case 1 + COMSTL_CCH_GUID:
        STLSOFT_CONTRACT_ENFORCE_POSTCONDITION
           _PARAMS_EXTERNAL(
           L'\0' == (*buff)[COMSTL_CCH_GUID],
           "buffer must be nul-terminated");
        break;
      default:
        if(S_OK == (hr = HRESULT_FROM_WIN32(
          STLSOFT_NS_GLOBAL(GetLastError)())))
        {
          hr = E_INVALIDARG;
        }
        break;
    }
  }
  return hr;
}
```

- Conditional parameter precondition contract enforcement
- *Postcondition* contract enforcement of the nul-terminating character of `*buff`.

`comstl_C_GUID_to_string_a()` (see Listing 11) is completely comprised of features we've already discussed. The only specific comment is regarding the use of the postcondition contract enforcement of the nul-terminating character, which `WideCharToMultiByte()` must return because we gave it the space in which to do so.

Ambient character encodings

Thankfully, that's it for all the long listings. Now we turn to the issue of selecting the requisite `_a`/`_w` function variant, dependent on our ambient character encoding. In C, this is just a matter of using macros, as in:

```
  #ifdef __cplusplus
  . . . /* The C++ functions */
  #else /* ? __cplusplus */

  # ifdef UNICODE
  #  define comstl_C_GUID_from_string
comstl_C_GUID_from_string_w
  #  define comstl_C_GUID_to_string
comstl_C_GUID_to_string_w
```

Listing 12

```
#ifdef __cplusplus
inline HRESULT comstl_C_GUID_from_string(
  wchar_t const* str
, GUID*          pguid
)
{
 return comstl_C_GUID_from_string_w(str, pguid);
}
inline HRESULT comstl_C_GUID_from_string(
  char const* str
, GUID*       pguid
)
{
 return comstl_C_GUID_from_string_a(str, pguid);
}
inline HRESULT comstl_C_GUID_to_string(
  REFGUID   guid
, wchar_t (*buff)[1 + COMSTL_CCH_GUID]
)
{
 return comstl_C_GUID_to_string_w(guid, buff);
}
inline HRESULT comstl_C_GUID_to_string(
  REFGUID  guid
, char   (*buff)[1 + COMSTL_CCH_GUID]
)
{
 return comstl_C_GUID_to_string_a(guid, buff);
}
#else /* ? __cplusplus */
. . . /* The C macros */
#endif /* __cplusplus */
```

```
  # else /* ? UNICODE */
  #  define comstl_C_GUID_from_string
comstl_C_GUID_from_string_a
  #  define comstl_C_GUID_to_string
comstl_C_GUID_to_string_a
  # endif /* UNICODE */

  #endif /* __cplusplus */
```

In C++, however, we prefer inline functions to macros. It would be a simple matter to do this, as in:

```
inline HRESULT comstl_C_GUID_from_string(
  TCHAR const* str
, GUID*        pguid
)
{
# ifdef UNICODE
 return comstl_C_GUID_from_string_w(str, pguid);
# else /* ? UNICODE */
 return comstl_C_GUID_from_string_a(str, pguid);
# endif /* UNICODE */
}
```

TCHAR is a Windows header-defined typedef, similarly conditional to either **wchar_t** or **char**, depending on the presence or absence of the UNICODE pre-processor symbol definition.

However, this is not the preferred option. In C++ we also have overloading available to us, which I use as shown in Listing 12.

For one thing, it supports C++ users who may wish to work with both character encodings in the same code-base. But there's a much more convincing reason, which we'll cover next.

Flexible C++ functions

Another C++ language feature is namespaces. Since, in C++ compilation units, all of the functions in the **GUID** façade suite reside within the **comstl** namespace, we can dispense with the **comstl_C_** prefix, allowing the user to decide whether they wish to explicitly qualify:

```
comstl::X();
```

or use a **using** declaration

```
using comstl::X;
X();
```

(They could also use **using** directives, but I'd strongly suggest they do not; see [2] for reasons against.)

So, we define a separate set of simplified-name function façades (Listing 13), implemented in terms of the ones already discussed. The points of note are:

- Overloaded to work with pointer and reference arguments
- Defaulting of the **comparisonSucceeded** parameter to **NULL**, in which case an exception is thrown
- Definition of **GUID_from_string()** as a function template, and the use of string access shims in its implementation to make it compatible with any 'shim'd type' (see [2] [3] [4] [11])

The first two points are straightforward, but the latter bears some explanation. The **stlsoft::c_str_ptr** string access shim is a function overload set whose purpose is to 'access' (involving either elicitation or conversion) the C-string form of a type. Arguments of extant overloads include **std::string**, **std::wstring**, **stlsoft::string_view**, **HWND**, **struct dirent**, and many more (including those defined, without any coupling to STLSoft, in other libraries and commercial projects). Anyone is free to define their own overloads to suit their projects. The precise return type from **stlsoft::c_str_ptr()** depends on the function arguments:

```
// in namespace stlsoft
char const* c_str_ptr(
  std::string const& s
);
wchar_t const* c_str_ptr(
  std::wstring const& s
);
```

Finally it's clear why both overloads (**char const*** and **wchar_t const***) are defined in C++ for **comstl_C_GUID_from_string()**: the compiler selects the overload of **stlsoft::c_str_ptr()** that matches **GUID_from_string()**'s **str** parameter, and then selects the overload of **comstl_C_GUID_from_string()** accordingly. This is one of the key elements of the application of string access shims in generalising APIs that can work with different character string encodings.

In the commercial project that motivated these façade functions, the file identifier string-forms reside within a memory mapped-file, and are accessed using string views [12] in the form of **stlsoft::string_view**. These instances are convertible to GUID via the requisite overload of **stlsoft::c_str_ptr()**.

Classes?

You understand, I hope, the motivation for providing façade functions as both C and C++ compatible. The final element in providing façades for the **GUID** type is to (re)use these façade functions in the implementation of façade classes, i.e. **comstl::guid**. In a future version of STLSoft 1.10, all previous uses of the Windows API **GUID<->**string functions will be replaced by the new ones described here. That will assist in my ongoing (and eternal, and unwinnable) battle to keep a clean separation in the levels of abstraction: 3rd-party API -> function façades -> class façades -> application code. ■

Acknowledgements

I'd like to thank Steve Love, the CVu editor, for dealing flexibly and patiently with this chronically deadline-slipped author, and my friends and review panel resident experts, Garth Lancaster and Chris Oldwood.

Listing 13

```
#ifdef __cplusplus
inline int GUID_compare(
  GUID const* lhs
, GUID const* rhs
, HRESULT* comparisonSucceeded = NULL
)
{
  HRESULT succeeded;
  int result = comstl_C_GUID_compare(lhs, rhs,
     &succeeded);
  if(S_OK != succeeded)
  {
    if(NULL == comparisonSucceeded)
    {
      throw com_exception("comparison failed",
         succeeded);
    }
    else
    {
      *comparisonSucceeded = succeeded;
    }
  }
  return result;
}
inline int GUID_compare(
  GUID const& lhs
, GUID const& rhs
, HRESULT* comparisonSucceeded = NULL
)
{
 return GUID_compare(&lhs, &rhs,
    comparisonSucceeded);
}
inline int GUID_binary_compare(
  GUID const* lhs
, GUID const* rhs
)
{
  return comstl_C_GUID_binary_compare(lhs, rhs);
}
inline bool GUID_equal(
  GUID const* lhs
, GUID const* rhs
)
{
 return 0 != comstl_C_GUID_equal(lhs, rhs);
}
inline bool GUID_equal(
  GUID const& lhs
, GUID const& rhs
)
{
 return 0 != comstl_C_GUID_equal(&lhs, &rhs);
}
template <typename S>
inline HRESULT GUID_from_string(
  S const& str
, GUID*    pguid
)
{
  return comstl_C_GUID_from_string(
     stlsoft::c_str_ptr(str), pguid);
}
inline HRESULT GUID_to_string(
  REFGUID   guid
, wchar_t (*buff)[1 + COMSTL_CCH_GUID]
)
```

Listing 13 (cont'd)

```
{
  return comstl_C_GUID_to_string_w(guid, buff);
}
inline HRESULT GUID_to_string(
  REFGUID  guid
, char    (*buff)[1 + COMSTL_CCH_GUID]
)
{
 return comstl_C_GUID_to_string_a(guid, buff);
}
#endif /* __cplusplus */
```

References

[1] 'Quality Matters, part 1: Introductions and Nomenclature', Matthew Wilson, *Overload* 92, August 2009

[2] *Imperfect C++*, Matthew Wilson, Addison-Wesley, 2004

[3] *Extended STL, volume 1: Collections and Iterators*, Matthew Wilson, Addison-Wesley, 2007

[4] 'An Introduction to FastFormat, part 2: Custom Argument and Sink Types', Matthew Wilson, *Overload* 89, April 2009

[5] 'An Introduction to FastFormat, part 1: The State of the Art', Matthew Wilson, *Overload* 88, February 2009

[6] 'Open-source Flexibility via Namespace Aliasing', Matthew Wilson, *C/C++ Users Journal*, July 2003

[7] *C Interfaces and Implementations*, David R. Hanson, Addison Wesley, 1997

[8] *Deep C Secrets*, Peter van der Linden, Prentice-Hall, 1994

[9] 'Safe and Efficient Error Information', Matthew Wilson, *CVu*, July 2009

[10] Null Object: Something for Nothing, Kevlin Henney, 2003 (originally presented at EuroPLoP in July 2002), available from http://www.two-sdg.demon.co.uk/curbralan/papers/europlop/NullObject.pdf

[11] 'An Introduction to FastFormat, part 3: Solving Real Problems, Quickly', Matthew Wilson *Overload* 90, June 2009

[12] 'A View To A String', Matthew Wilson, *Dr. Dobb's Journal*, January 2006

An Interview with Jerry Weinberg

Jon Jagger asks the questions.

Jerry Weinberg was born in 1933 and is the author of over 40 books and 400 articles on wide-ranging topics centred on software development. Jerry has also started writing science fiction novels and I can thoroughly recommend *First Stringers* and *The Aremac Project*. He is a winner of the J.D. Warnier prize in information science and the Stevens Award for Contributions to Software Engineering and was an inaugural inductee into the Computer Hall of Fame. As well as writing classics such as *The Psychology of Computer Programming*, *The Secrets of Consulting*, and *Quality Software Management volumes 1–4*, he is also a founder of the Amplify Your Effectiveness (AYE) conference and runs several world-class workshops including Problem Solving Leadership (PSL). I have read most of Jerry's books and had the honour of meeting him at the 2009 AYE conference and the 2010 PSL workshop.

If you were stranded on a desert island with five books which books would you choose and why? Please don't pick more than one of your own!

1. An empty notebook with as many square feet of blank paper possible, for keeping my journal of this desert island adventure.
2. The complete *Oxford English Dictionary*, so I can study the English language in depth for the future when I might be rescued.
3. Frazer's *Golden Bough*, so I can immerse myself in the vastness of human culture.
4. Martin Gardner's *The Annotated Alice* (Lewis Carroll's *Alice* books, annotated by Martin Gardner) so I could read for wisdom and humor at the same time. (I've used this book as a text in s/w development.)
5. If a Kindle counts as one book, I'd take that. Otherwise, *Wilderness Medicine, Beyond First Aid*, 5th Edition by William Forgey (I'd like to research this one, because I haven't read this, but I would need the best such book.)

I don't have any computer books on the list because I'm assuming I wouldn't have a computer.

If there were such a book, I'd probably want *How to Build a Two-Way Radio Out of Coconut Shells*.

Which books would you change and to what if you did have a computer?

1. I would not need the empty notebook, as I could keep my journal on-line. Instead, I'd want a complete service manual for the computer equipment.
2. I would not need the OED, for the same reason – an on-line OED. Instead, I would take Don Knuth's *The Art of Computer Programming*. I assume we count all published volumes as one 'book'.

The rest I would keep the same.

What do you consider your biggest contribution to the software world.

That's easy. I answered that some years ago, and my answer hasn't changed. My biggest contribution to the software world is that I never invented yet another programming language.

What would you still like to achieve?

I'd like to persuade more computer people to work on the unsolved problems of our profession, rather than the (pretty much) solved ones such as compiler writing.

Could you give some examples of what you consider to be the important unsolved problems of our profession

- Requirements: finding out what will really make people happy
- Doing the things we know we ought to do
- Not doing things we know we ought not do
- Conservation of knowledge from one generation to the next
- Developing some sense of standard practice (can be more than one, but not too many) that will be followed around the world

You've written that you get a lot of inspiration from nature. Could you give some examples – of both the inspiration and nature that inspired it.

- Systems grow, and when they grow, they change. I've learned a lot about growing systems from natural systems and how they grow. For instance, there's D'Arcy Thomson's book, *On Growth and Form*, and a lovely little book, *The Adaptive Geometry of Trees*.
- Systems adapt. Way back, I wrote *Natural Selection as Applied to Computers and Programs*, inspired by such works as Fisher's *The Mathematical Theory of Natural Selection*.
- Many of my insights from nature, along with references, can be found in my books on Systems Theory, such as *An Introduction to General Systems Theory* and *General Principles of System Design*.

If you had a one-time-only time machine how would you use it?

I wouldn't. I have a rule: Don't mess with time.

Suppose you used it to visit your younger self what advice would you give yourself?

When someone offers advice, you ought to taste it, but you don't have to swallow it.

JON JAGGER

Jon Jagger is a self-employed software coach-consultant-trainer-mentor-programmer who works on a no-win no-fee basis. He likes the technical aspects of software development but mostly enjoys working with people. He can be contacted at jon@jaggersoft.com

In question one you mentioned Martin Gardener's book, *The Annotated Alice*. Could you expand on how you used this as a text in s/w development.

Alice's trip across the chessboard to become promoted quite nicely parallels a typical development process. It's no coincidence that Lewis Carroll (Dodgson) was a mathematical logician. He was able to do logic, and to make memorable the instances of illogic. For example, the Red Queen's behaviour is that of many bad development managers ('Off with their heads.') The students were invited to find other parallels in the book, which hopefully set their minds to work.

What is the biggest change you'd like to see in the software world?

Slowing down in order to do things right.

What would it take to make this happen?

Hell would have to freeze.

What question would you ask yourself?

What question would you ask yourself?

And what is your answer.

What question would you ask yourself? Just kidding. That was a fun recursion.

(this is the real question Jerry would ask himself)

Why are you writing novels these days?

(and this is his real answer)

Like any life-changing decision, the switch to fiction has many reasons, all intertwingled. What follows are some of the reasons I have been able to disentangle.

All my life, I've dedicated myself to helping smart, talented people be happy and productive. You can see that theme in my books, I think, and it's the theme I've continued in my novels (see list below).

I have a rule. Don't mess with time!

But not all my work has been through writing. Dani and I have also spent our careers training these smart, talented people through the use of experiential workshops – Problem Solving Leadership Workshop (PSL), Organization Change Shop (OCS), Systems Engineering Management (SEM), and the Amplifying Your Effectiveness Conference (AYE). We use experiential training methods because they are effective. They reach many people, and much more deeply, than your typical lecture class with PowerPoint slides.

In many ways, reading a non-fiction book can be much like one of those PowerPoint lectures, so whenever possible, I have used stories to bring my non-fiction works to life. Stories have always been powerful for learning, going back thousands of years. Why? Because a good story arouses the readers feelings of participating in the experience the story describes.

A great deal of the popularity of my books (and other non-fiction writers like Tom DeMarco) is in the stories. They make for lighter reading, which some people love and some people find objectionable, but overall, I have managed to present lots of hard stuff effectively through these stories.

Some of my books have been directed specifically at Information Technology (IT) people. Some have not. Generally, the ones that have sold best and longest have been the ones not so specifically directed at IT people – books such as *An Introduction to General Systems Thinking*, *The Secrets of Consulting*, *Are Your Lights On?*, *What Did You Say?*, and *Weinberg on Writing: The Fieldstone Method*. These readers tell me they like the stories, even those that have some technical content. In fact, they often learn technical concepts and details as a byproduct of reading and enjoying the stories. I like that, because it says I have reached many smart, talented people who don't happen to be IT folk. The novels do that even better. I hope more people try them.

Could you briefly mention some of your favourite music and films.

Music: Almost anything Baroque. Any of Mozart (I have the complete recordings of his works, which should tell you something). Most music prior to 1850; almost nothing after that except Sousa, Gilbert & Sullivan, and Scott Joplin.

Films: (I'm probably missing some, but all these are favorites that I can see again at any time, with no hesitation.)

The African Queen (1951)
Bringing Up Baby (1938)
Casablanca (1942)
City Lights (1931)
Duck Soup (1933)
Fantasia (1940)
The Gold Rush (1925)
High Noon (1952)
It's A Wonderful Life (1946)
Modern Times (1936)
A Night At The Opera (1935)
Paths of Glory (1957)
Singin' In The Rain (1952)
The Treasure of the Sierra Madre (1948)
The Wizard of Oz (1939)

Could you expand a little on your rule 'not messing with time'? Does it relate to 'Slowing down in order to do things right'?

Yes. Things take the time they take, not the time you hope they will take. Pushing for half-time produces half-baked.

At your Problem Solving Leadership course I noticed you sometimes answered a question obliquely. For example when discussing cancer treatments you told the story of the Aspen mountain passes and how none of them are really much good. Could you explain why you sometimes choose to answer a question in this manner.

What you call oblique, I call powerful. Take your example. Few of my listeners have had cancer, but many of them have the experience of hiking on difficult trails. Thus, the lesson is more likely to stick, as it has with you.

You've said learning a new language (a human language, such as Spanish) really helped your ability to think in a Systems Thinking way. Could you expand on that?

In my case, it was French, learned while living in Geneva for a number of years. After a while, I found myself thinking in French when my ordinary thinking wasn't solving a problem. For instance, in English we say things like, 'I took an aspirin for my headache'. In French, the expression would be 'against my headache'. To me, that seems to imply something different from the English expression – a different way of thinking about disease and cure.

So, in effect, after learning French, I have a second way of addressing problems – and once I have a second way, it's easy to see that there may be a third way, and a fourth way, and so on. Being about to see a situation in multiple ways is a key concept in systems thinking, and learning a new language favors that concept.

The same phenomenon occurs in so-called programming languages. Someone who can work in only one such 'language' cannot truly be considered a real programmer, in my opinion. That's why in school, I always insisted that programming homework be done in at least two languages – along with a written analysis of how each language affected the way the student approached the problem. ■

Acknowledgement

Photograph courtesy of Fiona Charles.

A Game of Chase

Baron Muncharris plays a two-horse race.

Halloo Sir R-----! Pray come join me and partake of a glass of this rather excellent potation! Might I again tempt you with a wager? Splendid!

I have in mind a game that always reminds me of my victory on the turf at Newmarket. Ordinarily I would not participate in a public sporting event such as this since I am at heart a modest man and derive no pleasure in demonstrating my substantial superiority over my fellows.

On this occasion, however, I had been asked to ride by the Empress of Russia herself and could not honourably have refused. Finding herself in some temporary pecuniary difficulty she had wagered her jewellery on a gelding from her own stable and felt, with ample justification I might add, that my being his jockey would greatly improve her chances.

My steed seemed in good condition as we cantered to the starting line but, as soon as the starter dropped his flag, he stumbled and fell. As my fellow jockeys raced down the track I examined his leg and came to the conclusion that it had been deliberately injured; presumably by a bookmaker fearful of losses resulting from my inevitable victory.

Naturally I could not ride the injured animal, but nor could I entertain failing the Empress. I therefore took the beast upon my shoulders and started off at a sprint. It took fully half the circuit before I had caught up with the pack and another quarter before I had taken the lead, although in my defence I had not yet lunched.

That my victory was celebrated in fine style hardly bears telling. Rather, let us commence our game.

Here, I have placed your coin on the lower left hand square of this chessboard and mine on the lower right. At each turn I shall move my coin four squares away from yours around the perimeter of the board and you shall likewise move your coin towards mine, albeit by a number of squares determined by the roll of a die.

If my coin remains ahead of yours until it returns to the bottom row of the board I shall have your coin as my prize. If, however, you catch or pass my coin before I do so the chase shall end and you shall receive a bounty of forty coins for every square between mine and my goal!

When I described the rules of this game to that damnable student acquaintance of mine, he started tiresomely wittering on about his condition once more and about some convocation I can only suppose he is planning to attend. What interest he supposes a noble such as I might have in either the health or the calendar of a wretch such as he is quite beyond my comprehension.

Now take another draught and think on your chances! ■

(Listing 1 shows a C++ implementation of the game.)

Listing 1

```
unsigned
roll()
{
  return 1 + unsigned( 6.0 * double(rand())
     / (double(RAND_MAX)+1.0));
}
unsigned
play()
{
  unsigned b = 7;
  unsigned r = 0;
  while(b<28 && r<b)
  {
    b += 4;
    r += roll();
  }
  return (b<28) ? 40*(28-b) : 0;
}
```

On a Game of Guesswork

A student performs his analysis.

Recall that the Baron's game consisted of guessing under which of a pair of cups was to be found a token for a stake of 1 and 7/8^{ths} of a coin and a prize, if correct, of 1/8th of a coin. Upon success, Sir R----- could have elected to play again with three cups for the same stake and double the prize. Success at this and subsequent rounds gave him the opportunity to play another round for the same stake again with one more cup than the previous round and a prize equal to that of the previous round multiplied by its number of cups.

In other words, if Sir R----- had earned the right to play the nth round, he would have had to pay a stake of 1 and 7/8^{ths} of a coin to do so and must have guessed beneath which of n+1 cups was the token for a prize of:

$$\frac{1}{8} \times 1 \times 2 \times 3 \times \ldots \times n$$

At first glance the game seems to be heavily biased in the Baron's favour. Indeed, Sir R----- could only have come out ahead if he won 5 games in a row (see equation 1).

The chance of his having done so was a paltry

$$\frac{1}{2} \times \frac{1}{3} \times \frac{1}{4} \times \frac{1}{5} \times \frac{1}{6} = \frac{1}{720}$$

strongly suggesting he should not have taken up the Baron's challenge.

Appearances, however, can be deceptive. We can reckon Sir R-----'s expected winnings with the never-ending formula

$$-1\frac{7}{8} - \frac{1}{2} \times \left(\frac{1}{8} - 1\frac{7}{8} + \frac{1}{3} \times \left(2 \times \frac{1}{8} - 1\frac{7}{8} + \frac{1}{4} \times \left(2 \times 3 \times \frac{1}{8} - 1\frac{7}{8} + \frac{1}{5} \times \ldots\right.\right.\right.$$

Equation 1

$$\frac{1}{8} + 2 \times \frac{1}{8} + 2 \times 3 \times \frac{1}{8} + 2 \times 3 \times 4 \times \frac{1}{8} = 4\frac{1}{8} < 4 \times 1\frac{7}{8} = 7\frac{1}{2}$$

$$\frac{1}{8} + 2 \times \frac{1}{8} + 2 \times 3 \times \frac{1}{8} + 2 \times 3 \times 4 \times \frac{1}{8} + 2 \times 3 \times 4 \times 5 \times \frac{1}{8} = 19\frac{1}{8} > 5 \times 1\frac{7}{8} = 9\frac{3}{8}$$

Rearranging this yields

$$-1\frac{7}{8}-\frac{1}{2}\times 1\frac{7}{8}-\frac{1}{2}\times\frac{1}{3}\times 1\frac{7}{8}-\frac{1}{2}\times\frac{1}{3}\times\frac{1}{4}\times 1\frac{7}{8}-\ldots$$
$$+\frac{1}{2}\times\frac{1}{8}+\frac{1}{2}\times\frac{1}{3}\times 2\times\frac{1}{8}+\frac{1}{2}\times\frac{1}{3}\times\frac{1}{4}\times 2\times 3\times\frac{1}{8}+\ldots$$

and thus

$$-1\frac{7}{8}\times\left(1+\frac{1}{2}+\frac{1}{2\times 3}+\frac{1}{2\times 3\times 4}+\ldots\right)+\frac{1}{8}\times\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots\right)$$

Using an exclamation mark to denote the product of 1 and every integer greater than zero and less than or equal to that that immediate precedes it, known as the factorial, and a capital sigma to represent the sum of the expression that follows it with the variable term replaced one after the other by each of the integers between those below and above it, we have equation 2 (below).

The first sum is known as the exponential series and is equal to the mathematical constant e, or approximately 2.7183. The second sum is known as the harmonic series and is infinite in value! I expressed as much to the Baron, but I am not entirely sure he grasped my point.

This fact that the sum of these ever diminishing terms grows without limit seems unlikely, but it is relatively easy to demonstrate upon grouping them together. See equation 3.

The question that remains is how long must Sir R----- have been willing to play in order to exploit this advantage.

Unfortunately, I know of no tidy formula with which to express the sum of the first n terms of the harmonic series. We can, however, figure bounds on it using the integral calculus.

The sum of the first n terms of this series is trivially equal to the area under the curve that takes the value of 1 between 0 and 1, of 1/2 between 1 and 2, of 1/3 between 2 and 3 and so on and so forth up to $1/n$ between n-1 and n. Now this curve is everywhere above that of $1/(x+1)$ and nowhere above that of the lesser of 1 and $1/x$.

Using the integral calculus to calculate the areas under these two bounding curves we can conclude that

$$\int_0^n \frac{1}{x+1}dx < \sum_{i=1}^{n}\frac{1}{i} \le 1+\int_1^n \frac{1}{x}dx$$

and hence, by the rules of integration

$$\ln(n+1) < \sum_{i=1}^{n}\frac{1}{i} \le \ln n$$

If we treat the first n terms of the exponential series as if it were equal to its entirety, a reasonable supposition if n is not small, Sir R-----'s expected winnings if he elects to never play more than n games, E_n, would have been approximately bounded by

$$1\frac{3}{4}-1\frac{7}{8}\times e+\frac{1}{8}\times\ln(n+1) < E_n \le 1\frac{3}{4}-1\frac{7}{8}\times e+\frac{1}{8}\times(1+\ln(n))$$

Now the number of games that Sir R----- must have been prepared to play if he wished to come out ahead on average is bounded by

$$1\frac{3}{4}-1\frac{7}{8}\times e+\frac{1}{8}\times\ln(n+1) < 0 \le 1\frac{3}{4}-1\frac{7}{8}\times e+\frac{1}{8}\times(1+\ln(n))$$

which, upon rearranging, gives

$$14-15\times e+\ln(n+1) < 0 \le 15-15\times e+\ln(n)$$

and therefore, for large n, approximate bounds of

$$15\times e-15 \le \ln(n) < 15\times e-14$$
$$25.7742 \le \ln(n) < 26.7742$$
$$e^{25.7742} \le n \le e^{26.7742}$$

or, in more familiar notation, roughly

$$156{,}000{,}000{,}000 < n < 425{,}000{,}000{,}000$$

I should consequently have advised Sir R----- to take up the Baron's challenge, but only if he had a taste for a very large prize at very long odds and plenty of time on his hands! ■

Figure 1 shows bounds for the harmonic series.

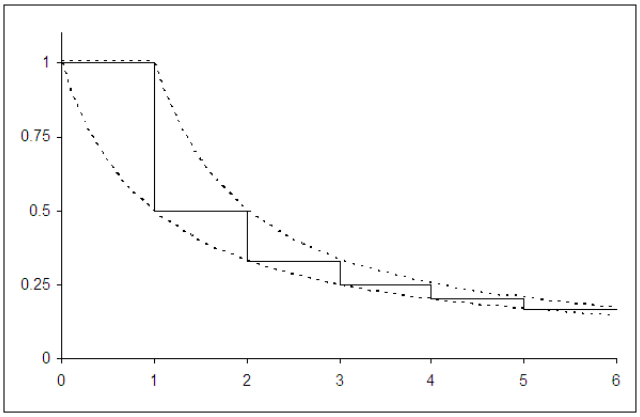
Equation 2

$$-1\frac{7}{8}\times\sum_{I=1}^{\infty}\frac{1}{n!}+\frac{1}{8}\times\sum_{i=2}^{\infty}\frac{1}{i}=-1\frac{7}{8}\times\left(\left(\sum_{i=0}^{\infty}\frac{1}{i!}\right)-1\right)+\frac{1}{8}\times\left(\left(\sum_{i=1}^{\infty}\frac{1}{i}\right)-1\right)$$
$$=1\frac{3}{4}-1\frac{7}{8}\times\sum_{i=0}^{\infty}\frac{1}{i!}+\frac{1}{8}\times\sum_{i=1}^{\infty}\frac{1}{i}$$

Equation 3

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}+\frac{1}{9}+\frac{1}{10}+\frac{1}{11}+\frac{1}{12}+\frac{1}{13}+\frac{1}{14}+\frac{1}{15}+\frac{1}{16}+\ldots$$
$$=1+\frac{1}{2}+\left(\frac{1}{3}+\frac{1}{4}\right)+\left(\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8}\right)+\left(\frac{1}{9}+\frac{1}{10}+\frac{1}{11}+\frac{1}{12}+\frac{1}{13}+\frac{1}{14}+\frac{1}{15}+\frac{1}{16}\right)+\ldots$$
$$>1+\frac{1}{2}+\left(\frac{1}{4}+\frac{1}{4}\right)+\left(\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}\right)+\left(\frac{1}{16}+\frac{1}{16}+\frac{1}{16}+\frac{1}{16}+\frac{1}{16}+\frac{1}{16}+\frac{1}{16}+\frac{1}{16}\right)+\ldots$$
$$=1+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\frac{1}{2}+\ldots$$

Figure 1



AN APOLOGY

A footnote was unintentionally omitted from 'On a Game of Nerve' (CVu 22.3) that identified Louis Lavery as Monsieur L.... and thanked him for his solution.

Code Critique Competition 65

Set and collated by Roger Orr.

Please note that participation in this competition is open to all members, whether novice or expert. Readers are also encouraged to comment on published entries, and to supply their own possible code samples for the competition (in any common programming language) to scc@accu.org. A book prize is awarded to the winning entry.

Last issue's code

I'm trying to write a simple quadratic equation solver for the equation "a * x * x + b * x + c = 0" that writes output to a file but I am having problems getting it working. It's OK for some inputs but I'm having problems, in particular with equations that have no (real) solution. The code is in Listing 1.

Listing 1

```
C:> cc64
Enter quadratic coeffs: 1 0 -1
Roots: 1 and -1
C:> cc64
Enter quadratic coeffs: 1 2 3
Roots: -1.#IND and -1.#IND
Didn't match

// Solve quadratic equation.
// Save the roots.
// Read and verify they wrote OK.

#include <cmath>
#include <fstream>
#include <iostream>
void tofile(double a, double b)
{
   std::ofstream("file.dat") << a << b;
}
void fromFile(double & a, double & b)
{
   std::ifstream("file.dat") >> a >> b;
}
void verify(double rootHigh, double rootLow)
{
  double readRootHigh, readRootLow;
  fromFile(readRootHigh, readRootLow);
  if ((readRootHigh != rootHigh) ||
      (readRootLow != rootLow))
  {
     std::cout << "Didn't match" << std::endl;
  }
}
int main()
{
  std::cout <<
    "Enter quadratic coefficients: ";
  double a,b,c,rootHigh,rootLow;
  if (std::cin >> a >> b >> c)
  {
    rootHigh = (-b + sqrt(b*b - 4*a*c)) / 2*a;
    rootLow = (-b - sqrt(b*b - 4*a*c)) / 2*a;
    std::cout << "Roots: " << rootHigh <<
      " and " << rootLow << std::endl;
    tofile(rootHigh, rootLow);
    verify(rootHigh, rootLow);
  }
}
```

Critiques

Thaddaeus Frogley <codemonkey.uk@gmail.com>

The program as presented has two maths problems:

1. if b*b < 4*a*c then it will try to take the square root of a negative number.
2. if a == 0 then it will divide by zero

Both of these operations are invalid with normal floating point (or integer) maths.

Assuming solutions involving complex numbers[1] are outside the scope of the assignment, the student should modify his code to explicitly check for these cases and present the user with an appropriate message instead of a garbage result.

Recommended reading for the student:

http://en.wikipedia.org/wiki/Floating_point

Specific attention should be paid to section 4.2, which talks about Special Values, and will explain the meaning of "-1.#IND", and section 7 which talks about exceptional cases.

References:

[1] http://en.wikipedia.org/wiki/Complex_number

Richard Polton <drboots@galactichq.org>

This is possible within C++ using the `std::complex` data type:

```
#include <complex>
#include <iostream>

int main( int, char**)
{
  double a=1, b=2, c=3;

  std::complex<double> x1 = (-b + sqrt(
    std::complex<double>(b*b - 4*a*c)))
    / ( 2*a);
  std::complex<double> x2 = (-b - sqrt(
    std::complex<double>(b*b - 4*a*c)))
    / ( 2*a);

  std::cout << "the two roots are: " << x1 <<
    " and " << x2 << std::endl;
  return 0;
}
```

However my programming language du jour is F# so I thought a little function to do the same thing using the equivalent library function wouldn't go amiss...

```
#r @"c:\FSharp\FSharpPowerPack-
2.0.0.0\bin\FSharp.PowerPack.dll";;
open Microsoft.FSharp.Math;;
let toComplex = fun x -> Complex.mkRect( x, \
  0.0);;
```

ROGER ORR

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

```
let quadraticSolver (a:float) (b:float) \
(c:float) =
    let sqrt = (fun x -> (Complex.mkRect \
      (x,0.0))) >> Complex.sqrt
    (toComplex(-b) + sqrt(b*b - \
      4.0*a*c))/toComplex(2.0*a),
    (toComplex(-b) - sqrt(b*b - \
      4.0*a*c))/toComplex(2.0*a);;
```

[Editor's note: I've used a trailing `\` to show the split lines]

Sadly, I couldn't work out how to extend the **+** et al. operators to have dissimilar types, e.g. a float and a complex number, so I had to stick with the helper function **toComplex**. Overloading the **sqrt** function was much easier :-)

Huw Lewis <huw.lewis@hlsoftware.co.uk>

First pass

Serialisation

There is a problem with the serialisation scheme used. This writes two **double**s to the file as text. As there is nothing between the two numbers to delimit them, they cannot be reliably read back e.g: the numbers 1.2 and 3.4 would be serialised as: "1.23.4". So, on reading the first number we get 1.23, then 0.4 for the next.

The solution for this is simple – add white-space between the two numbers.

Operator precedence

The calculation of the quadratic equation roots has errors with operator precedence. I can remember from my old school days that the solution for a quadratic equation is:

```
roots = ( -b +/- sqrt( b2 - 4ac) ) / 2a
```

However, the code as given ended with:

```
/ 2 * a;
```

This would have the effect of dividing by 2, then multiplying the result by 'a', which is not the desired effect. Amend to the intended:

```
/ (2 * a);
```

Other observations

Main function is declared with an **int** return type but has no **return** statement.

Now that we have a return code, it should indicate success only if the operation succeeds – including the verification of the serialised results.

The **verify** method uses the **!=** operator for the **double** type. This is notoriously troublesome as all sorts of rounding errors and precision issues can result in a tiny difference between values that are intended to be the same. Instead, use **<** and **>** operators with an acceptable error margin.

rootHigh and **rootLow** variables are declared in a scope where they aren't really needed. Prefer to declare variables within the scope they are required.

Quadratic equations

Quadratic equations are of the form:

$0 = ax2 + bx + c$

The solution to quadratic equations is:

$rootHigh = (-b + sqrt(b2 - 4ac)) / 2a$

$rootLow = (-b - sqrt((b2 - 4ac)) / 2a$

Note there are two solutions (roots) to the equation and these may be real (a simple number) or complex (a 2-D vector consisting of real and imaginary components).

So, the code as-is will support only real roots, complex roots will manifest themselves as good old nan (not a number) results.

Complex numbers

Complex numbers are a concept used to handle some (normally) impossible operations such as the square root of a negative number. They are a 2-D vector consisting of a real component and an imaginary component, *i* where *i* represents $-1^{1/2}$.

Lucky for us, C++ provides a Complex Number type in the STL **<complex>** header. This is a template that takes a type argument for the underlying real/imaginary components.

So, the equation will have complex roots if the expression to be 'square rooted' is negative:

b2 - 4ac

When the above expression is negative, we know that it represents the square of an imaginary component of the complex number.

sqrt(b2 - 4ac) = i * sqrt (abs(b2 - 4ac))

So, we can calculate our complex root (high) as follows:

```
complex<double> rootHigh = -b;
rootHigh.imag(sqrt(abs(b * b - 4 * a * c)));
// imaginary component
rootHigh /= (2.0 * a);
```

The root (low) calculation is as above but with the imaginary component being applied in the negative.

Tidying up

Now we have used complex numbers, we have some maintaining to do.

It is necessary to handle complex number types in a few places e.g. the calculations, the serialisation, verification etc. Therefore it would be convenient to define a **typedef** for our **std::complex<double>** to aid readability: **ComplexNumber_t**.

The **tofile**, **fromFile** and **verify** functions should now operate on **ComplexNumber_t** rather than simple **double**s. The white-space added earlier to delimit the simple **double**s is no longer required.

In order to use the streaming operators from **<complex>** the serialisation lines must be split to separate the construction from the streaming operations.

It was found that by default, the streaming of **double** types was printing only to 6 significant figures causing verification errors when comparing with the calculated values to many more decimal places.

- Increase the precision to 10 significant figures
- Calculate the acceptable error margin as the least significant of the 10 sig figures.

Solution

```
// Solve the quadratic equation.
// Save the roots.
// Read and verify they wrote ok.

#include <cmath>
#include <fstream>
#include <iostream>
#include <complex>
namespace // anon for file-scope
{
  // Define a complex number type for this program
  typedef std::complex<double> ComplexNumber_t;
  // Define some constants
  enum{
    double_precision = 10 // significant figures
  };


  // Write the given complex number results to a
  // file: file.dat
  void tofile(const ComplexNumber_t& a,
     const ComplexNumber_t& b)
  {
```

```
    std::ofstream output("file.dat");
    output.precision(double_precision);
    output << a << b;
  }

  // Read the complex number results from
  // the file.dat
  void fromFile(ComplexNumber_t& a,
     ComplexNumber_t& b)
  {
    std::ifstream input("file.dat");
    input >> a >> b;
  }

  // return the number of digits in the
  // whole number part of the double
  // precision number.
  int decimal_order( double n )
  {
    return static_cast<int>( std::log10(n) ) + 1;
  }
  bool doubles_are_equal( double a, double b )
  {
    // Get the difference between the two doubles
    const double diff = std::abs( a - b );
    if ( diff == 0.0 )
    {
      // a and b really are equal
      return true;
    }
    else
    {
      // there is a difference, but how big is it?
      double min_var = std::min( std::abs(a),
         std::abs(b) );

      // define the error margin as the least
      // significant digit
      const double error_margin =
         decimal_order(min_var) / std::pow( 10,
         double_precision - 1);
      return (diff < error_margin);
    }
  }
  bool verify(const ComplexNumber_t& rootHigh,
     const ComplexNumber_t& rootLow)
  {
    // read back the serialised results
    ComplexNumber_t readRootHigh, readRootLow;
    fromFile(readRootHigh, readRootLow);
    return ( doubles_are_equal(
        readRootHigh.real(), rootHigh.real() ) &&
      doubles_are_equal( readRootHigh.imag(),
        rootHigh.imag() ) &&
      doubles_are_equal( readRootLow.real(),
        rootLow.real() ) &&
      doubles_are_equal( readRootLow.imag(),
        rootLow.imag() ) );
  }
} // end namespace

int main()
{
  // define return code constants
  enum
  {
    ret_ok = 0,
    ret_error = 1
  };
  int retCode = ret_error; // assume error until
                           // operation completes
  // Get the inputs from the user
  std::cout <<
      "Enter quadratic coefficients: ";
  double a, b, c;

  if (std::cin >> a >> b >> c)
  {
    // Declare the roots as complex numbers
    ComplexNumber_t rootHigh;
    ComplexNumber_t rootLow;

    // Quadratic Equation solution:
    // ( -b +/- sqrt( b*b - 4*a*c ) ) / ( 2*a )

    // evaluate the fragment in the square root
    double sqrtExpression =
      ( b * b ) - ( 4 * a * c );
    if (sqrtExpression < 0.0)
    {
      // the roots will be complex.
      // make the sqrtExpression positive,
      // then assign to the imaginary
      // component.
      sqrtExpression = std::sqrt(
        std::abs(sqrtExpression) );
      rootHigh = -b;
      rootHigh.imag(sqrtExpression);
      rootHigh /= (2.0 * a);
      rootLow = -b;
      rootLow.imag( sqrtExpression * (-1.0) );
      rootLow /= (2.0 * a);
      std::cout << "Roots are complex: "
        << rootHigh << " and "
        << rootLow << std::endl;
    }
    else
    {
      // The roots will be real
      sqrtExpression =
        std::sqrt(sqrtExpression);
      rootHigh = (-b + sqrtExpression) /
        ( 2.0 * a );
      rootLow = (-b - sqrtExpression) /
        ( 2.0 * a );
      std::cout << "Roots are real: "
        << rootHigh.real() << " and "
        << rootLow.real() << std::endl;
    }
    // Serialise results
    tofile(rootHigh, rootLow);
    // verify the result serialisation
    if (verify(rootHigh, rootLow))
    {
      // Operation succeeded
      retCode = ret_ok;
    }
    else
    {
      std::cout << "Verification failed." <<
        std::endl;
    }
  }
  return retCode;
}
```

Commentary

The code sample has several problems. Firstly, the code is broken, as Huw points out, because the operator precedence rules mean the code doesn't do what was expected: `expression / 2*a` ignores white space and

Inspirational (P)articles

Richard Polton discovers that a little thought and insight can save a lot of time.

I have recently discovered Project Euler (http://www.projecteuler.net) and had elected to work through the problems as part of my teach-myself-F# course. One of the problems involved counting the number of factors in the ascending sums of the natural numbers. Although I had previously tackled a similar problem concerning prime factorisation, I made a small error.

My program was successful but, curiously to me at the time, took over two weeks solid processing time to arrive at the answer. This seemed wrong to me and so while the program was running I attempted a number of optimisations in an effort to find the mistake. Most mysterious!

The issue came about because of the way in which the factors were determined. I used a brute force algorithm, trial division, but forgot that the upper limit should be **`sqrt n`** and not **`n`** itself. Therefore, each iteration was taking a significantly longer time! In fact, whe n I changed the upper limit to **`sqrt n`** then the program took a mere seven seconds to complete! That's a certain improvement over two weeks!

Code Critique Competition (continued)

binds the operators left to right; so it is equivalent to **`(expression / 2) * a`**.

Secondly there are various problems serialising double precision numbers to a file – the C++ iostream metaphor seems broken to me when, as in this case, **`operator<<`** and **`operator>>`** calls are not in general reciprocal. Note that changing the data type to **`std::complex<>`** solves this issue since the output format in this case is bracketed – but I suspect this is a fragile solution.

Thirdly there is the question over what answer is actually desired when the mathematical solution is a complex number. Generalising the calculation to return values of this type may be the best solution, but of course the format of `file.dat` has to change in this case.

Finally there is the vexed question of infinity. Unfortunately the C++ is silent on the string that will be output when a floating point value is infinite (or a Nan) and equally silent on whether or not such a value can be streamed into a double.

A helper class or similar can be used to assist with this particular problem if a solution is needed; if not then the program should be changed to return an error where a non-representable result is returned.

The winner of CC 64

There are three very different critiques above. The first two provide ways to avoid the problem either by explicitly failing for some inputs or solving a more general problem. The last critique engages more with the original problem (and also pointed out the problem with the operator precedence) so I am awarding this issue's prize to Huw.

Code Critique 65

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

As mentioned in the last issue Louis Lavery had a proposal for a minor change in the formulation of the code critique. His proposal goes 'State a problem together with a solution and ask for improvements (which, in some cases, might lead to a full rewrite due to taking a different view or to generalising and in others to the conclusion there's no way to improve it)'.

The problem: Print all pairs of indices that bracket the maxima in a circular list of numbers.

For example:

```
0 1 2 3 4 5 6 7 8 9 <- indices
2,1,4,3,5,6,6,6,1,1 <- numbers
^   ^     ^ ^ ^     <- maximums (three 6s
                       count as one).
(9,1) (1,3) (4,8)   <- brackets (output in any
                       order).
```

No doubt this can be made more efficient. Simply, after we print a pair we could set j to k, couldn't we? Well, we'd have to detect the case where k's been reset to zero and, if so, return immediately. I guess k's only going to be reset half the time, so we'll be setting a **`k_was_reset_flag`** to no avail half the time, so is it really more efficient? It certainly makes the code more complicated – less easy to follow.

Maybe it's better to write two functions, one brackets the maximums in a non-circular list and the other looks for a wrapped maximum?

The code is shown in Listing 2. You can also get the current problem from the accu-general mail list (next entry is posted around the last issue's deadline) or from the ACCU website (http://www.accu.org/journals/). This particularly helps overseas members who typically get the magazine much later than members in the UK and Europe.

Listing 2

```
// The code to improve
void bmax(int const a[],int const n)
{
  int i = n-1;
  for(int j = 0; j != n; ++j)
  {
    if (a[i] < a[j])
    {
      int k = j;
      do
      {
        if (++k == n) k = 0;
      } while(a[j] == a[k]);
      if (a[j] > a[k]) print(i,k);
    }
    i = j;
  }
}
```

Desert Island Books

Chris Oldwood chooses his companions to take to the island.

Although I have seen Chris around at conferences and other ACCU events and on accu-general, I had never put the name to the face until the night he claims he put me off my stride. What he fails to mention about that night was the huge compliment he paid me in the pub afterwards.

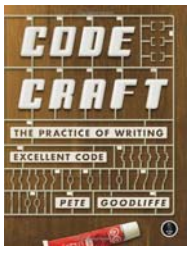
When Chris sent me his desert island books I thanked him on receipt before reading them and he warned me to save my thanks until I read them. When you read them you'll see that they are very good, but I couldn't help pointing out that as the column editor I'd have the final say! With that in mind I'd like to point out that my respect for Chris has taken a slight dip. Not only did he mock Spandau Ballet, but he clearly feels that *Borderline* is a better song than *Like A Prayer*. Utter madness.

Chris Oldwood

I have a strange feeling that I know why I might be stuck on this desert island – it's probably penance for knocking the host of this column off his stride whilst he was giving a presentation to the London branch of the ACCU. Unlike the heckling from the Henney corner that night mine was a little more subtle. I forgot my glasses and so kept squinting at the slides in what must have seemed to be a 'very confused' manner – actually maybe it's not penance but sabotage…

So where do I start? I guess I'll follow the pattern of my predecessors and generally cover the technical books first. The music should be easy, but the novel is going to be pretty tough as I just don't read fiction books. As a child I'd have preferred a complete set of Haynes manuals to the collective works of William Shakespeare; and as I saunter into my technical library (aka The Downstairs Toilet) I find myself staring at the modern equivalent – umpteen books on [D]COM and Windows' internals. On a pragmatic note I feel the most useful would be four of the books by Jeffrey Richter as he has produced some paper based behemoths that could easily be bound together to make a pretty sizeable raft.

Hmmm, I thought this was going to be easy. There aren't many unfinished books on the shelf (although I'm ashamed to admit that one is by our very own Pete Goodliffe) so perhaps I should just take those? Looking back at previous choices I seem to have some serious omissions. I've been writing C++ for about 15 years and yet I don't own a copy of Bjarne Stroustrup's *The C++ Programming Language* or Nicolai Josuttis' *The C++ Standard Library*. Come to think of it I don't own a copy of the standard either; in fact I don't think I've ever seen one; then again I've never suffered from insomnia.



Anyway I probably shouldn't use this time to learn about a specific technology but perhaps use it to reflect on our profession instead – I'm sure I spend way too much time 'doing' and not nearly enough thinking about 'how' I'm doing it. And so we come full circle and Pete Goodliffe's *Code Craft* comes back off the shelf again. I've read the first 75 or so pages and it looks like a modern take on the software development process so I reckon it's a good start. My 1st Edition of *Code Complete* by Steve McConnell is looking a little outdated so perhaps this will supplement it nicely. Plus it comes with monkeys, although I suspect I won't be short of those for company on the island.

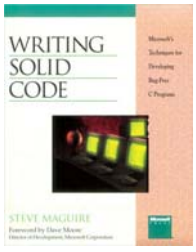
A couple of years ago I raised a question with my fellow ACCU colleagues about what modern books there are on the Object Orientated paradigm. I realised I had stumbled into C++ development and consequently the OO world without really understanding the principles of the paradigm I was supposed to be following. It seems as though the age old tomes by Grady Booch (*Object Oriented Analysis* and *Design with Applications*) and Bertrand Meyer (*Object-oriented Software Construction*) are still a force

to be reckoned with even now. Hubert Matthews chipped in with the more recent *Object-Oriented Design Heuristics* by Arthur J. Riel. I've browsed through that and it looks very useful, especially with the slightly different format, but I think I'll go with the Meyer classic as it covers the basics. It's also another hefty beast and so should come in handy to weigh down any tarpaulin.

In terms of shelf space Kent Beck barely registers (he creates very svelte works by comparison) and yet I've found his books an absolute joy to read. He has a pleasant conversational style that almost makes you feel like he's in the room reading to you (now that would make a novel episode of Jackanory). *Implementation Patterns* in particular is one of those books that really tries to help you get inside the author's head. Some might say much of it is obvious, but that's only because he's pointed it out. If there was a *Collective Works of Kent Beck* available to satisfy these ham-fisted rules I'd take that; after all I only read *Extreme Programming Explained* because I enjoyed reading *Implementation Patterns* and *Test Driven Development*. So let's see what else Amazon says he has to offer, back in a moment…

Oh, that's it really. But that's good because now I have another choice that I wasn't expecting. Whilst on Amazon I checked my ever growing wish list to see what else might inspire me. One subject leaps out and that is to better appreciate what it is that my manager is [supposed to be] doing. I digested Steve Maguire's *Debugging the Development Process* a very long time ago when I was but a junior starting out. Although I thoroughly enjoyed it I'm not convinced I would have read it with this goal in mind. *Peopleware: Productive Projects and Teams* by Tom DeMarco and Timothy Lister appears to be the classic text everyone refers to and so I'd probably go with that as my starter. Perhaps when I've been rescued I could delve a little deeper and try something more modern – I seem to remember that Allan Kelly bloke having a book out.

So I'm down to my last choice and I'm feeling somewhat guilty. I've only scratched the surface of *The Pragmatic Programmer* (Andrew Hunt and David Thomas) even though it is considered an all time classic and I know it's packed to the gills with sound advice. I've also just started *Coders at Work* (Peter Seibel) as the whole 'Duct Tape Programmer' debacle caught my attention. It's a collection of interviews with such famous luminaries as Donald Knuth. It would be neat if I could use that as my novel-come-not-so-hardcore-techie-book because it's a bit more touchy-feely instead. So when I said there aren't many unfinished books on my shelf what planet was I on? What would be a really bold move is to pick one of those books Jon Jagger was selling at the ACCU Conference this year; there was some seriously weird looking titles in there and yet so intriguing at the same time. This software business is way deeper than you think…



Actually I've known all along the one book I was definitely going to take and that is *Writing Solid Code* by Steve Maguire. This (along with *Code Complete* by Steve McConnell which is arguably a significantly more useful book) had a huge impact on me and the team I joined back at the start of my career. I know nostalgia isn't what it used to be, but I still have a great fondness for this book – most notably the anecdotes from Steve's days managing projects at Microsoft. I personally found the airing of their dirty laundry particularly refreshing; it added a sense of humility and real-worldliness

to it. As the books subtitle suggests (*Microsoft Techniques for Developing Bug-free C Programs*) it has less direct relevance to me these days but I still enjoy thumbing through the pages of this (and his sister book *Debugging the Development Process*) to read random anecdotes for the occasional sanity check. Blast, now I've realised I don't have room for Raymond Chen's *The Old New Thing*, and after I gave it such a glowing review in C Vu as well!

So with my geeky side nourished I come back to the thorny issue of what novel to take. It's probably apparent from my earlier comment that I've barely read any novels as it's not a pastime I indulge in. So do I go for the safety of what little I already know or pick something I think I'll like? There's probably going to be a beach on the desert so I feel inclined to pick one of those 'best sellers' by Jackie Collins or Andy McNab. Not really my scene though. If I can't abuse the rules and get *Coders at Work* accepted then maybe I should try for *The Cuckoo's Egg* by Clifford Stoll. This is about how he tracked down a hacker back in the 80's whilst working at the Lawrence Berkeley Labs. For a techie I thought he penned a pretty good book.

Now if it were films life would be easier with anything from Sergio Leone's westerns to Michael Bay's glossy sci-fi flicks to choose from. Here's a novel thought (ha ha), perhaps I should read the book of a film I enjoy – everyone always says how much better they are. That would probably push me towards some sort of Philip K Dick affair, but I'd rather go with something humorous. Douglas Adams vs Terry Pratchett; it's so predictable. Oh, alright then, it has to be *The Hitchhiker's Guide to the Galaxy*.

Is anyone still reading at this point? Have I managed to portray myself as such a shallow character by now that you've decided I probably deserve to be marooned on a desert island to reduce pollution of the gene pool? Good, then I shall hit you with my musical taste… Let me start by pointing out that I'm a product of the 1980s. It's ok, I'm not going to pick Spandau Ballet, but I do covet the keyboard and sampler rather than guitar so anyone looking for more weight to the Purple Floyd vs Deep Pink or whatever the argument was can go back to sleep now.

My love of beeps and squeals starts with Kraftwerk, passes through Depeche Mode on to The Prodigy, The Chemical Brothers and ends with a variety of DJs such as Carl Cox. There is also the occasional diversion into ska, hip hop and rap. I suspect that a turntable would fair better than a CD player in such a dusty climate, but sadly the vast majority of my vinyl collection is in the form of 12" singles so I would have to pick something like the *Soft Cell 12" Singles Box Set* to get value for money out of my musical choices. With only two options available it would need to be a pair of those rarest of albums; the ones that you could listen to relentlessly and don't have a single duff track that you always skip. There is plenty of house and techno that I listen to relentlessly at work as I find it an enjoyable accompaniment; the disruptive exceptions being *Music for the Jilted Generation* by The Prodigy and *Swordfish (The Album)* by Paul Oakenfold. These have a habit of forcing me to context switch in search of the 'track repeat' button. If it wasn't for my IBM Model M keyboard I suspect Voodoo People would have had to answer for quite a few broken keys by now.

For 'The Island' where solitude is likely the order of the day, I'd prefer lyrics and big beats so I'm going to pick *Confessions on a Dance Floor* by Madonna and *To The 5 Boroughs* by The Beastie Boys. Madonna has teased us in the past with a few cracking tracks on *Ray of Light and Music* (and in secret I'll even admit to adoring *Borderline* from the '80s) but *Confessions* is 100% perfect pop with a modern dance/house edge. The Beastie Boys on the other hand ticks all the boxes I'm looking for in the hip hop/rap/humour department. It was a close call between *Hello Nasty* (their previous best work) and *Boroughs* but the later has an extra level of maturity and polish. Depending on whether or not my family are also marooned with me I could allow for a significant increase in the swearing ratio and pick *Encore* by Eminem instead.

Let me finish with my own J. J. Abrams inspired addition to the format by asking 'What type of clunky old terminal would you like to find in the bunker hidden under the island?' For me it would be an RM 380Z as that's where it all started.

What's it all about?

Desert Island Disks is one of Radio 4's most popular and enduring programmes. The format is simple: each week a guest is invited to choose the eight records they would take with them to a desert island (http://www.bbc.co.uk/radio4/factual/desertislanddiscs.shtml).

The format of 'Desert Island Books' is *slightly* different from the Radio 4 show. You choose about five books, one of which must be a novel, and up to two albums. Some people even throw in the odd film. Quite a few ACCUers have chosen their Desert Island Books to date and there are plenty more to go.

The rules aren't too strict but the programming books must have made a big impact on your programming life or be ones that you would take to a desert island. The inclusion of a novel and a couple of albums helps us to learn a little more about you. The ACCU has some amazing personalities and Desert Island Books has proved we only scratch the surface most of the time.

Each issue of CVu will have someone different. If you would like to share your Desert Island Books please email me: paul.grenyer@gmail.com.

CONGRATULATIONS STEVE AND FRANCES

The CVu team, along with the rest of the ACCU membership, would like to send our congratulations and best wishes to the CVu editor Steve Love and Inspirational (P)articles author Frances Buontempo on their recent marriage.

We are glad to know that your honeymoon can only interrupt magazine production the one time!

It's slightly worrying that Frances is now offically called Dr Love. We're not sure if she's become an evil Bond villain or a daytime television agony aunt.

We wish you many happy years together.

Bookcase

The latest roundup of book reviews.

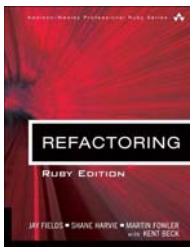
If you want to review a book, your first port of call should be the members section of the ACCU website, which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

After you've made your choice, email me and if the book checks out on my database, you can have it. I will instruct you from there. Remember though, if the book review is such a stinker as to be awarded the most un-glamourous 'not recommended' rating, you are entitled to another book completely free.

I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

Jez Higgins (jez@jezuk.co.uk)

Refactoring, Ruby Edition



By Jay Fields, Shane Harvie, Martin Fowler with Kent Beck, published by Addison Wesley, ISBN: 978-0-321-60350-0

Reviewed by Gavin Heavyside

Fowler and Beck's original *Refactoring* book is deservedly a classic, and helped to formalise a set of best-practices that should be second nature to all aspiring software craftsmen. This version updates the original to use Ruby examples and idioms throughout.

The book begins with a simple worked example that illustrates how a series of small independent refactorings can lead to radically different – and better – program structure while preserving all outward behaviour. It then covers the principles underpinning refactoring, and catalogues code 'smells' that warn the programmer of areas that might deserve refactoring.

The 'Smells' section details several warning signs that your code needs work, and developing an intuitive sense of smell for code is the key to applying refactorings successfully. Conventional warning signs like duplicated code are highlighted, but also smells that are more Ruby oriented such as 'Metaprogramming Madness'. Metaprogramming is a powerful feature of Ruby and powers some common Ruby libraries, but it can easily lead to obfuscated code that is impossible to decipher.

The bulk of the book comprises a catalogue of named refactorings. Each refactoring is named, described, and the motivation behind its use is given. The mechanics of performing the refactoring are given in a step-by-step description, and a simple example of performing the refactoring is given in Ruby. The catalogue covers a wide range of refactorings, both updated versions of the original text, and new ones that take advantage of Ruby language features (e.g. 'Replace Loop with Collection Closure Method'). There are some pairs of complementary refactorings (e.g. 'Change Reference to Value' and 'Change Value to Reference') that acknowledge all changes are context-dependent.

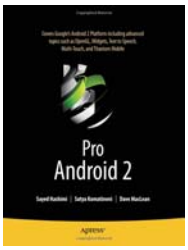
The scope of refactorings cover a diverse range topics and scopes, from the trivial ('Rename Method') all the way up to more wide-ranging architectural changes ('Convert Procedural Design to Objects'). In each case the reasons why you would consider the refactoring are covered, followed by how to do it. Class diagrams accompany most of the refactorings, which is perhaps the first time I've ever seen UML used to describe Ruby code.

Refactoring is the process of changing the structure of a program, without affecting its behaviour. I've lost count of the times I've heard developers claim to be 'refactoring' some code without any unit tests, and if you don't have tests then there is no reliable verification of its behaviour. Without tests, you aren't refactoring, you are just rewriting. The book includes a short (9 page) introduction to unit testing with the `Test::Unit` framework, but it cannot provide the background needed to develop a test suite sufficient to perform major refactoring.

In my experience TDD is fairly pervasive in the serious Ruby community, and every self-respecting Rubyist uses either a combination of rspec and cucumber or some of the other leading test frameworks. The concepts and language from the original Refactoring book have permeated modern test-driven practices, and it seems to me that this familiarity has somehow diluted the impact that this edition ought to have.

It is a good book with much to recommend it, particularly the catalogue of refactorings, but for me it slightly falls between two stools. If you are have unit test coverage to the point where you are ready to refactor, there is a good chance you are already doing some of the things that this book describes. If not, then you need to bootstrap your TDD. If you are new to the world of TDD and refactoring then it should be read in conjunction with a good TDD book for you to get the most benefit.

Pro Android 2



By Sayed Hashimi, Satya Komatineni, and Dave MacLean, published by Apress, ISBN : 978-1430226598

Reviewed by Derek M Jones

This book deals with writing applications to run under Google's Android 2 platform/operating system/API. It is Java specific and those wanting to use C or another language will have to look elsewhere. The book has a strong hands-on feel to it with the authors sprinkling the text with useful practical tips and warnings.

Like any modern platform Android contains a huge amount of functionality and even a book of 736 pages can only skim the surface. As might be expected in a book about a platform from Google there is a strong bias towards applications that use the internet and search.

The user interface APIs include support for OpenGL, widgets, text to speech and touchscreen. The material strikes a good balance between providing an introduction to the topic and practicalities of using the API. In places the quantity of code is excessive, but in general it is well integrated into the discussion of the topic at hand.

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 me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please let us know so they can be added to the list

- **Holborn Books Ltd** (020 7831 0022) www.holbornbooks.co.uk
- **Blackwell's Bookshop**, Oxford (01865 792792) blackwells.extra@blackwell.co.uk

View From The Chair

Hubert Matthews
chair@accu.org

As the rest of us drift through the summer months, our conference committee despite their important day jobs are flexing their not-insubstantial brains and preparing for another blockbuster conference. The conference has grown to be a large part of the ACCU and its public face. If it continues to grow then we will again outgrow a venue, so the time has come to start thinking about what happens when we do. There has been quite a bit of internal discussion on the matter already and I think that it time that this discussion encompassed the membership as a whole. There are lots of points to take into consideration: which city, the range of type and price of accommodation, whether accommodation across multiple venues is acceptable, easy access to pubs and restaurants, whether the main venue has a bar, etc. There are, of course, pros and cons to each of these and different people will have potentially radically different opinions. Some factors we can't control (such as getting time off work or companies to pay the costs, long-distance travel) but some we can control and we need to understand the effect of possible changes. For instance, what can we do to encourage people who have never come to the conference to come for the first time. Instead of trying to collate the opinions of the most vocal majority by asking on accu-general, we are considering using a web-based survey to solicit everyone's views. Moving conference venues isn't something we want to do lightly so your input and views are, as ever, important; watch this space.

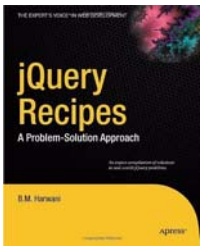
Gathering information on people's views is something that I think we should be doing on a wider scale within the ACCU and not just for the conference. The ACCU has a large geographical spread with a wide range of members, ranging from professional developers and consultants – some with international reputations – through to keen amateurs and those who do not earn their living through developing software. Some people we know well from the conference and some through local groups. Others we know hardly at all. I would like to conduct a membership-wide survey to see what people would like from the ACCU and where they think it should be heading. At the basic level, do people want more local groups? What would encourage them to turn up, or even better help organise a group? Do they want speakers or is a pub meeting adequate? How can the ACCU help? We could then start to explore larger issues about what defines success for the ACCU, how and in what direction we want to expand (if at all) and what we think the ACCU stands for. We have grown organically up till now and perhaps this is the time to take stock and discuss what we want for the future. Retrospectives are popular with agile developers so why not with us? I'm sure we have between us a whole raft of ideas, thoughts and wisdom kicking in the back of our minds but it hasn't been brought out or collated. The ACCU is a volunteer organisation and will therefore only thrive with the participation and engagement of members of all levels and types, rather than just the well-known and vocal ones. Our sense of community is one of the things that defines us – let's build on it and provide something for everyone.

Bookcase (continued)

While obtaining and using system resources is covered, it is not covered in any detail and a developer wanting to make sophisticated use of system resources will need to look elsewhere.

I get the feeling that Android 2 is still evolving and some of the information in the book is liable to change. However, developers have to start somewhere and this book covers the basic functionality, provides useful hands-on information and is readable (it does not attempt to provide exhaustive coverage and complete reference information is freely available elsewhere). In 12-18 months time things may have moved on sufficiently that a new book is required, in the meantime this book is recommended.

jQuery Recipes: A Problem-Solution Approach



By B. M. Harwani, published byApress, ISBN 978-1-4302-2709-0

Reviewed by Alan Lenton

Cookbook/Recipe style books seem to be all the rage at the moment. Although I do prefer reference books, I do find the cookbook style useful for things that I don't do very often, and this book was no exception. I would emphasize, though, that it's not a book you could really use to learn how to use the library.

jQuery is one of the most widely used JavaScript libraries and the book provides solutions to a wide range of the problems you are likely to encounter. The books starts out with the basics - selecting and using the DOM, and moves on to more complex situations from there. I found the form validation examples, and the event handling material particularly useful.

Each entry consists of statement of the problem, followed by a solution, and then a longer or shorter explanation and discussion of how and why the solution works. The stuff I used out of the book worked just fine, with no errors. Obviously, I didn't use everything, but the quality of the code provided seemed fine to me.

I was, however, a bit disappointed by the quality of the book production. The paper it's printed on is rather low quality, and some of the fonts used in displaying sample output are reproduced in very small type, making it difficult to read. Overall the level of graphic design leaves something to be desired. Fortunately, the content manages to overcome this handicap.

I found it useful, but I suspect this is partly a matter of taste. Ten years ago I would have recommended dropping into your nearest computer bookshop and browsing through this book and the O'Reilly equivalent to see which one is more to your taste. Sadly the dominance of Amazon has ended the possibility of this sort of activity, as well as the possibility of serendipity in the finding of books you never knew existed.

ERRATUM

An incorrect URL was given for the Software Toolkit in Ian Bruntlett's 'Competency Scale' article in CVu 22.3. The correct URL is: http://contactmorpeth.wikispaces.com/SoftwareToolkit

Delivering technical software to developers for more than 20 years with world-class service

QBS Software is a responsive Microsoft licensing partner with dedicated account managers who are Microsoft Licensing Sales Specialists. They understand Microsoft licensing options fully and will listen to your requirements and provide clear information on all of the options that suit your circumstances.

It does not matter which software you require; at QBS Software you will always find someone responsible to speak to. There are no automated telephone systems and no hard-sell. QBS seeks to develop a transparent partnership with you to help you to obtain the service and software that fits your needs best.

Below are just a few of the respected publishers and products that you can find on our web site:

Get Visual Studio 2010 with MSDN and start building outstanding applications

Microsoft Visual Studio 2010 Professional

- Silverlight tools
- Sharepoint development
- Cloud development
- Web development
- Generate from usage
- Multi-core development
- Windows development
- Office development
- Customisable IDE
- New WPF editor.

Microsoft Visual Studio 2010 Premium

- UI Test automation
- Performance profiling
- Code coverage
- Database change management
- Database unit testing
- Test impact analysis
- Static code analysis
- Code metrics
- Test data generation
- Database deployment.

Microsoft Visual Studio 2010 Ultimate

- IntelliTrace
- UML modelling
- Architecture explorer
- Logical class designer
- Load testing
- Layer diagram
- Test Manager 2010
  - Test case management
  - Manual testing
  - Web testing.

Microsoft Visual Studio Test Professional 2010

Simplify test planning and manual test execution using a specialized toolset that enables effective collaboration between developers and testers.

Microsoft Visual Studio Team Lab Management 2010

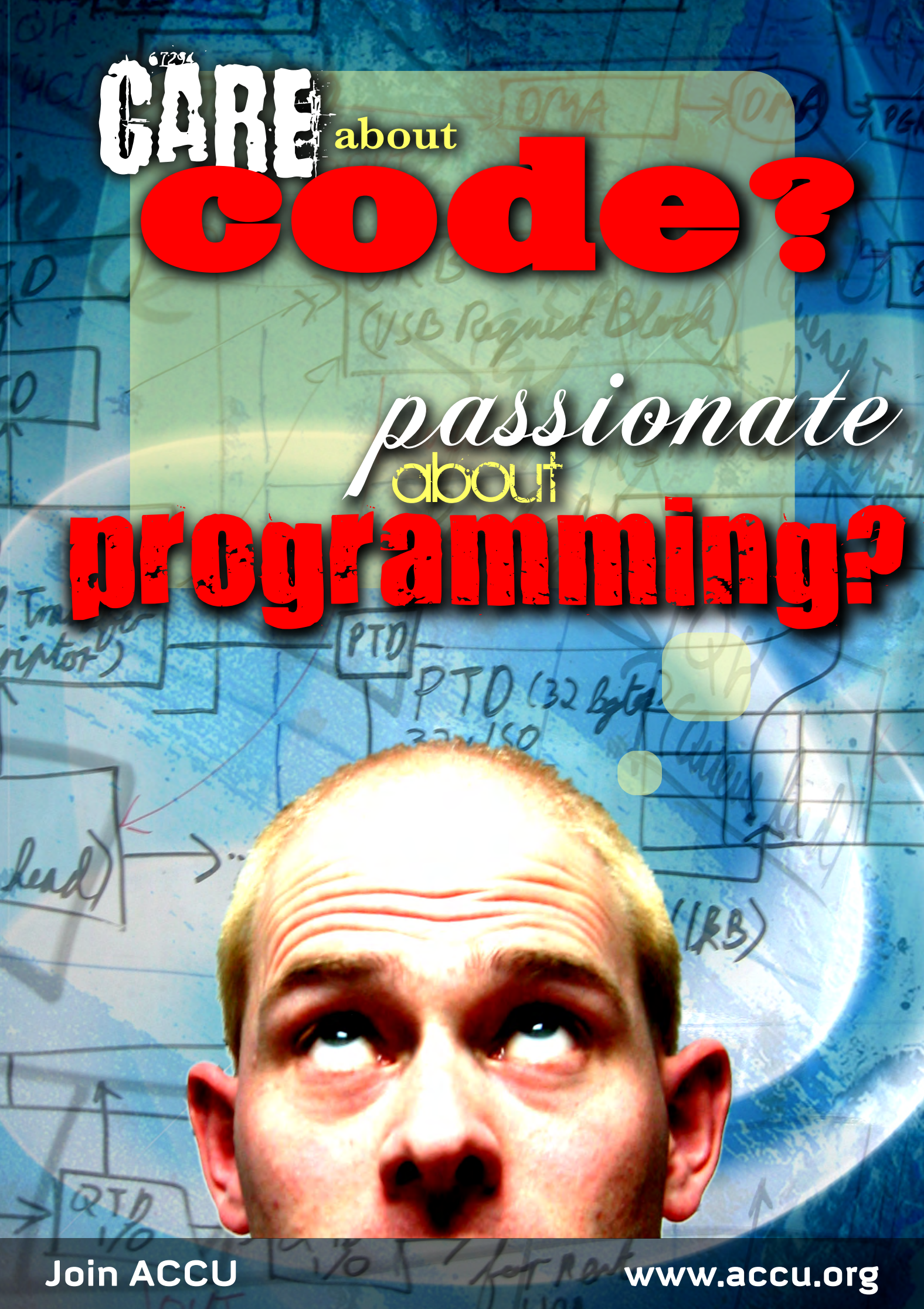
Create and manage virtual environments on a pool of Hyper-V hosts and System Center Virtual Machine Manager library servers.

Why Call QBS?

Microsoft GOLD CERTIFIED Partner

QBS is a Microsoft Gold Certified Partner specialising in providing licensing solutions for decades. All QBS Account Managers are Microsoft Licensing Sales Expert accredited and will provide clear information on all the options that suit your circumstances.

Please Call QBS today on 08456 580 580.


CARE about code?
passionate about programming?
Join ACCU
www.accu.org