The magazine of the ACCU

{cvu}

Volume 25 • Issue 1 • March 2013 • £3

Features

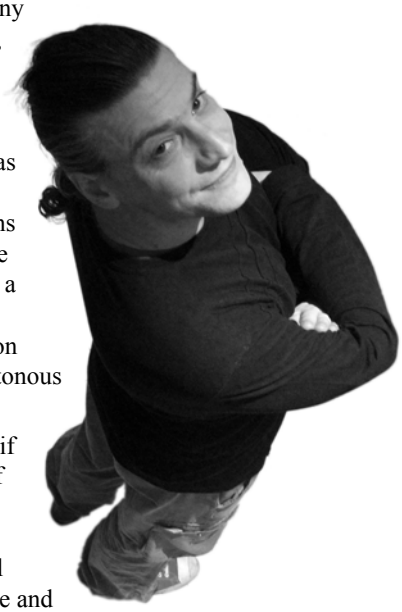
Regulars

{cvu}
Volume 25 Issue 1
March 2013
ISSN 1354-3164
www.accu.org

accu

Slave to the Grind

When Chris Oldwood sent me the article about his Toolbox, printed in this edition of *C Vu*, it brought to mind a few things. If any of you have seen Pete Goodliffe give a presentation, you'll know he's a *big* fan of tools and automating the donkey-work of day-to-day development: things like running tests, building the code, generating test data. The list is a very long one. In fact, there's a confluence of similar ideas throughout this edition with Joana Simoes writing about creating help files, and Richard Polton testing various versions of an algorithm, along with Chris' introduction to the tools he uses in support of the activity of creating software. Tools are a vital part of what we, as software developers, do all the time. Compilers, interpreters, linkers, loaders, editors, IDEs, version control systems are all tools for automating repetetive, monotonous and error-prone activities.



Pete's refrain, and a concept picked up by Chris here, is that if you find yourself doing the same thing more than a couple of times, automate it. If necessary, by writing your *own* tool. It might be a simple script or batch file, or a full-blown application, as necessary for the task. If there's already a tool available for what you need, all the better. If it's Open Source and you can get involved in its development, better still.

Sometimes the use of tools can be a hindrance, though, and this was the 'other' thing of which I was reminded. Modern IDEs such as Eclipse and Visual Studio provide – at least the plug-in facility for – many productivity tools. The ability for an editor to prompt you with the names of member functions of an object, or local variables isn't really new, but it did prompt an erstwhile colleague of mine to coin a new meaning for TDD: Tools Driven Development. In Visual Studio especially, another co-worker described C# programming as 'dot programming', referring to Intellisense™ popping up a list of members in response to typing `.` after a variable name.

It's important for lots of reasons that we own the tools, rather than the other way around. Which is why I still like programming in a 'simple' text editor, and building my code from a command line. It at least gives me the illusion that I am still master of the toolchain, rather than slave to it.

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 25.2: 1st April 2013
C Vu 25.3: 1st June 2013

Overload 115: 1st May 2013
Overload 116: 1st July 2013

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

Help!

Joana Simoes demonstrates some features of the Qt framework for help files.

Documentation is probably, invariably, the 'weakest link' in the software production chain. Last minute releases do not leave a lot of time for documenting the code, and excuses such as 'the code is self-explanatory' are more than common. Except that it is not. Probably you will not understand why you were using that `typedef` with a `multimap` inside a `multimap` two months later; and so nobody will.

But I'll probably leave this topic for another article (which is really worth it!) and just assume for now, that your code is thoroughly commented all the way, and that you even adopted an automated documentation generator (like Doxygen [1]) and formatted your comments accordingly. So my article is not going to be about you, or the other programmers using your application, but about the users. Yes, that's right, that class of living beings that potentially possess no technical knowledge, and can be the 'source' of all sorts of problems. Unfortunately, these may be the ultimate target of your program, and they don't care so much about the 'beauty' of templates, one-line conditional return expressions, or initialisations on the constructor, as they care about being able to understand and actually use the program. And this is why they need 'help'.

Background

For a long time, help was not really part of any API, and when implemented it was in some sort of proprietary format, done from scratch in the application (Basic, Java, Fortran and other structured languages). With the advent of object oriented languages, a format for help arose in Windows, and it was supported until Windows Vista: WinHelp format [2]. After that, fortunately Microsoft started pushing towards an HTML-based help, which is much more standard since it can be implemented across platforms; this is the only help format supported in the latest versions of Visual Studio. Normally help is not implemented by the same programmers that wrote the code, but by a different set of people, which probably explains the existence of a lot of third-party tools for writing help files (helpStudio Lite [3] is an example).

As a C++ programmer, producing cross-platform applications, I am an enthusiastic adopter of the QT libraries [4]. Qt, also pronounced 'Cute' provides an extensive framework for diverse things including (but not limited to): UI, multi-threading, STL, XML, and Help. Finding the Qt Help Framework [5] was a good surprise, due to its capabilities, neat integration, ease of use, and mostly for its ability to involve non C++ programmers in the process of creation of Help. I was personally involved in the process of creating a Help for an application with a non-programmer person, and apart from some small misunderstandings, I can say that the whole process was quite smooth. Thus I decided to share this tool with the rest of the ACCU users.

Anatomy of the Cute 'Help'

We can view a help project as a help manual, and a help collection as a set of manuals that may have cross references between them (for instance an application suite). All these project files are registered using Qt tools, and ultimately they generate a compressed file, which is read by Qt Assistant.

The 'Help project' basically consists of one or more HTML files describing many aspects of the application, and a project file that organises them (`.qhp`). The project file has information on 'how-to' use these files, organising the table of contents, the filter and the keywords. This is basically an XML file, such as the one in Listing 1.

In this file, you may notice important content definitions such as: 'filters', 'table of contents' and 'keywords'.

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>
<QtHelpProject version="1.0">
<namespace>mycompany.com.myapplication.1_0<namespace>
<virtualFolder>doc</virtualFolder>
<customFilter name="My Application 1.0">
 <filterAttribute>myapp</filterAttribute>
 <filterAttribute>1.0</filterAttribute>
</customFilter>
<filterSection>
 <filterAttribute>myapp</filterAttribute>
 <filterAttribute>1.0</filterAttribute>
 <toc>
  <section title="My Application Manual" ref="index.html">
  <section title="Chapter 1" ref="doc.html#chapter1"/>
  <section title="Chapter 2" ref="doc.html#chapter2"/>
  <section title="Chapter 3" ref="doc.html#chapter3"/> </section>
 </toc>
 <keywords>
  <keyword name="foo" id="MyApplication::foo" ref="doc.html#foo"/>
  <keyword name="bar" ref="doc.html#bar"/>
  <keyword id="MyApplication::foobar" ref="doc.html#foobar"/>
 </keywords>
 <files>
  <file>classic.css</file>
  <file>*.html</file>
 </files>
</filterSection>
</QtHelpProject>
```


JOANA SIMOES

Joana is an AGILE software developer for the Food and Agriculture Organisation (UN), who dreams of becoming a comic artist one day. Contact her by email at doublebyte@gmail.com, or in person in Park Guell (Barcelona) at lunchtime

Listing 2

```
<?xml version="1.0" encoding="UTF-8"?>
<QHelpCollectionProject version="1.0">
<docFiles>
 <register>
  <file>doc.qch</file>
 </register>
</docFiles>
</QHelpCollectionProject>
```

Listing 3

```
...
<docFiles>
 <generate>
  <file>
   <input>doc.qhp</input>
   <output>doc.qch</output>
  </file>
 </generate>
 <register>
  <file>doc.qch</file>
 </register>
</docFiles>
...
```

As both XML and HTML are well-known and text-based formats, we can create these files using any tool we like, ranging from text editors (Kate, Vi, Emacs) to XML or HTML editors, in case you are more 'visual' (normally non-programmers, such as the people who write manuals, are!).

Because we store help as compressed files, rather than raw files, the next step is to transform the qhp file into its compressed format (qch). Fortunately, the Qt suite comes with a CLI for that: the qhelpgenerator. This is an example of the syntax:

```
qhelpgenerator doc.qhp -o doc.qch
```

A 'Help collection' is composed of a set of help projects and a collection file (`.qhcp`). The top-level project that pulls together one, or many, help projects is called a Help collection; the collection is organised by a collection file (`.qhcp`), which is again, another XML file. In the current example, we have one help collection with our single project (see Listing 2).

And to generate the compressed collection file (qhc), we use another CLI: the qcollectiongenerator. This is an example of the syntax:

```
qcollectiongenerator mycollection.qhcp
   -o mycollection.qhc
```

The Help collection file is the output that is going to be called by the application. If we want to generate it in one-pass (skipping the qhelpgenerator step), we just have to modify the collection file slightly, like Listing 3.

Writing the manuals

The Help manual can be written in one or many html pages. There is no need to create a table of contents (toc), since the help will generate one on the navigation tree, on the left panel of Qt Assistant. However, its a good practise to have a face page, with a presentation of the manual and a table of contents. In this sample project, we have two files: index.htm, the face page that is displayed in the opening of the help and test.htm, with the actual help contents. To be able to refer to sections of the html file in the toc of the project file, and to be able to recall keywords, we need to create "anchors" to certain parts of the document. We can do that by enclosing some words with the <a> tag. For instance: if we want to create an anchor for the keyword "logbooks" in the file test.htm, we would write:

```
<a name="logbooks">logbooks</a>
```

and later recall this location with the url:

```
test.htm#logbooks
```

Writing the project file

In the project file we define the following attributes of the help project:

- Namespace: a unique identifier for the documentation set.
- Virtual folder: root directory of all files referenced in a compressed help file. In this way we can have relative paths shared between help collections.
- Custom filters: a list of attributes that allow us to display documentation only for them.
- filters: this section assigns contents to the defined custom filters.

Inside the 'filter' section we have the most important part of the help file: the toc, the keywords and files. The toc defines the table of contents of the help that is going to be displayed on the left panel of the QT assistant. In terms of syntax, we have a nested list of titles that allows us to define the hierarchical structure of the manual. For instance in the section of the project file shown in Listing 4, we recreate a structure like similar to Figure 1.

Figure 1

```
* Introduction
** Nature of the Information Supported by Myprog
** The Sampling Process
** Rational of Data Hierarchy
* Store Aliens Data
** Creating the Sampling frame
*** Create a new frame from scratch
**** Create a new frame by choosing an existing one
*** Characterize Sampling Technique
* Define Minor Strata
```

Listing 4

```
<toc>
<section title="Myprog Manual" ref="index.htm">
 <section title="Chapter 1: Introduction" ref="test.htm#chapter1">
  <section title="Nature of the Information Supported by Myprog" ref="test.htm#nature_information"/>
  <section title="The Sampling Process" ref="test.htm#sampling_process"/>
  <section title="Rational of Data Hierarchy" ref="test.htm#rational"/>
 </section>
 <section title="Chapter 2: Store Aliens Data" ref="test.htm#chapter2">
  <section title="Creating the Sampling frame" ref="test.htm#create_sampling_frame">
   <section title="Create a new frame from scratch" ref="test.htm#frame_from_scratch">
    <section title="Create a new frame by choosing an existing one" ref="test.htm#frame_from_existing"/>
   </section>
   <section title="Characterise Sampling Technique" ref="test.htm#sampling_technique"/>
  </section>
 </section>
 <section title="Define Minor Strata" ref="test.htm#minor_strata"/>
</section>
</toc>
```

Note that the name of the section is associated with a piece of an html document, by referring to an anchor that we defined in the html file. For instance in this example:

```
<section title="Chapter 2: Store Aliens Data"
 ref="test.htm#chapter2">
```

We associate the section 'Chapter 2: Store Aliens data' to the anchor 'chapter2' that was defined in `test.htm`.

The section 'keywords' is where we define the keywords that we can use to search for contents in the help index. For instance in the following line we associate the keyword 'logbooks' with a section of the file `test.htm` where we defined an anchor. The id, is an identifier that lets us refer this in the code (for instance creating a context sensitive help).

```
<keyword name="logbooks" id="Myprog::logbooks"
  ref="test.htm#logbooks"/>
```

Finally, in the 'files' section we register all the files that we want to include in the help project (in this case only two html files).

Viewing the Help

During the production stage, the Help team may not have access to a running application to view the Help. That is not a problem, since we can view the help files using the standalone Help viewer from Qt: the Qt Assistant. Assuming you do not have it already, you just have to install the Qt Sdk from Digia and call the Qt Assistant (on Windows go to the Start menu, and choose QTSDKx > Tools > QtAssistant). To register the manual go to Edit > Preferences > Documentation and add the path to the qhc file. If everything goes well, the index file and the table of contents should be displayed on the left panel; the contents will be displayed on the right panel. [6]

Figure 2 is a screenshot of the application Help, running on Qt Assistant.

Some (very important) things to keep in mind when writing the help

This section highlights some 'mistakes' that the help team I was working with faced. I learned that even some things that seem obvious to programmers, may not be obvious at all to people who are not used to dealing with technical constraints (e.g.: syntax); this is not a problem, as long as we are aware of them and describe everything properly.

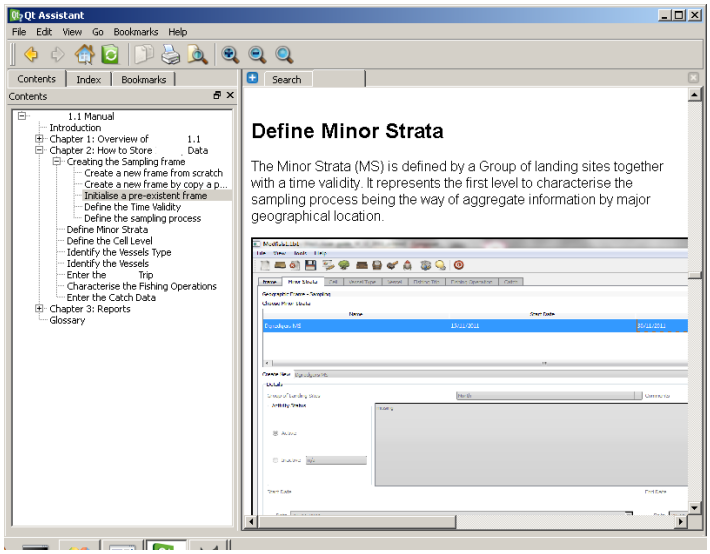
Startup page

It is advisable to have a startup page, to work as a 'face' of the help collection. In our example, this page (`index.htm`) is located in this url:

```
qthelp:myprog.app.1_1b/doc/index.htm
```

In this file, we display a small introduction and the toc. On the table of contents of the project file, the section that contains the face page, should be the outer section: i.e., the one that contains all the other ones!

Figure 2




```
<section title="Myprog 1.1b Manual"
   ref="index.htm">
  ....
</section>
```

Consistency of titles across project

The titles of the different sections should be consistent across the project. Make sure to double check the titles on:

- table of contents (in our example in `index.htm`)
- throughout the html document
- in the definition of the index of the qhp file.

In this example I opted to remove the word 'Chapter' from everywhere, except the structure on the qhp file. When using chapter numbers, choose a numbering system and remember to keep using it consistently, i.e., romans, literals, etc.

Consistency of structure across project

The same thing applies to the structure of the help: make sure to keep the same nested structure across table of contents, help html file and help definition file.

Creating the name tags

The name tags should enclose the section, since they contain the contents that we want to reference. For instance this is correct:

```
<a class="mozTocH3" name="mozTocId527307">
Create a new frame by copy a pre-existent one</a>
```

and this is not correct! (although it works, we are referencing empty content before the section title)

```
<a class="mozTocH3" name="mozTocId527307">
</a>Create a new frame by copy a pre-existent one
```

It is advisable to use human-friendly tags, to easily assimilate the references through code and avoid mistakes. For instance to reference 'Create a new frame by copy a pre-existent one', we may use the **`name="pre-existent"`** instead of **`name="mozTocId527307"`**!

Keywords

The Keywords section in the qhp file is very important, as it allows us to use the search function on the index, and also to create a context sensitive help in the UI. Please allow some time to define a few important keywords in the content.

Paths on the qhp file (IMPORTANT!)

The paths on the qhp file are the references that allow the user to navigate through the help contents; therefore it is very important for the help project that they are correct! On the path:

```
"user_guide_30_12_2011.html#logbooks"
```

We have two parts.

```
"user_guide_30_12_2011.html"
```

is the name of the html file: make sure this file exists and the name is spelled correctly (including the extension)!

```
"logbooks"
```

is the part after the **#** and it references a tag, which means somewhere in the html file we must have something like this:

```
<a name="logbooks">logbooks</a>
```

You need to ensure this exists. Take some time to make sure each one of the entries in the **<toc>** section of the qhp file is valid!

```
<section title="Introduction"
 ref="user_guide_30_12_2011.html#mozTocId386761"/>
<section title="Chapter 1: Overview of Myprog 1.1"
 ref="user_guide_30_12_2011.html#mozTocId42934">
....
```

The Downs and Ups of Being an ACCU Member

Chris Oldwood reflects on what the ACCU's ever done for him.

I joined the ACCU about 6 years ago. At the time I was working at one of the large investment banks and was mourning the recent loss of the *C/C++ Users Journal* on the company's chat system. I much prefer paper based journals and, what with the demise of *C++ Report* and *Windows Developer Journal* in previous years too, my rucksack was getting lighter on my daily commute. Naturally the channel I posted on was about C/C++ and it was suggested that I take a look at the ACCU. Little did I know there was also a dedicated ACCU channel on that chat system too…

maybe I didn't know quite as much as perhaps I thought I did

If someone had asked me in an interview, before joining the ACCU, how I rated myself as a programmer, out of ten, I'd probably have gone for a 7. I got a good start to my professional career working with smart people and in a culture where continuous learning was promoted. Various journals were circulated, as were books like *Code Complete* and *Writing Solid Code*. After leaving the company to go freelance I quickly realised I missed the journals and books and so took out my own subscriptions and started to build up my own library.

By the time I had joined the ACCU I felt pretty confident that I knew what I was doing. I had worked with plenty of people and at a few different organisations so felt I could gauge roughly where I sat on this hypothetical 'programmer scale'. My focus had always been on learning the technology as that what was interested me most – my bookshelf was mostly filled with the collective works of Jeffrey Richter. Although I had been 'doing OO and C++' for the last 10 years I actually didn't own any of the fundamental C++ texts, such as Stroustrup or Josuttis, instead favouring the *Essential* and *Exceptional* book series as a paper form of Lint.

And then I started going to the monthly ACCU London meetings and tagging along to the lunches. There I got to put faces to some of the names in the chat window and meet some of the other members that had graced the pages of the journals I had started reading. Up to that point most of the articles and books I had read were written by US based authors, but now I was starting to come face-to-face with these people. That was scary, and yet obviously cool at the same time. What started to become apparent was that maybe I didn't know quite as much as perhaps I thought I did. One particular conversation stands out as a watershed moment.

I had just been to see Paul Grenyer do his 'Boiler Plating Database Resource Cleanup' talk for ACCU London. As was the natural order of

CHRIS OLDWOOD

Chris started as a bedroom coder in the 80s, writing assember on 8-bit micros. Now it's C++ and C# on WIndows. He is the commentator for the Godmanchester Gala Day Duck Race and can be reached at gort@cix.co.uk or @chrisoldwood

Help! (continued)

The same applies to the **<keywords>** section of this file: make sure all the keyword references exist and are valid; the same applies to the TOC on file `index.html`.

Registering html files

At the end of the qhp file, make sure all the html files contained in this project are correctly listed, in order to be registered by the compiler. In this project:

```
<files>
  <file>index.htm</file>
  <file>user_guide_30_12_2011.html</file>
</files>
```

All the files used in the help project must be included in the **<file>** tag: this includes all the image files (jpg, png, etc)!

Conclusions

Overall I think the Qt Help framework presents an easy solution that makes the creation of help a less tedious task. The separation between the help and the application makes it possible:

- to develop and test the help, without having the source code of the application.
- to implement the help without having any knowledge of programming (although a basic knowledge of markups, such as XML and HTML would be nice!).

However, we do not want the help to be completely separated from the application. In fact, there is a part of the help that is linked closely to the application: what we call context sensitive help. This part requires implementing calls to the help engine on the code, and I thought I would leave it to another article.

Finally, as a 'goodie' for using Qt, the help is also completely portable across platforms.

See you soon

See you soon with more articles about the Qt framework. You can email me to suggest specific parts of Qt you would like to see covered, or to beg me to stop writing more articles! ■

References and notes

[1] Doxygen. Doxygen, http://www.stack.nl/dimitri/doxygen/

[2] wikipedia. winhelp, https://en.wikipedia.org/wiki/WinHelp

[3] Innovasys. HelpStudio Lite, http://www.innovasys.com/products/hslite/overview.aspx?cpid=vssdk

[4] Digia. Qt, http://qt.digia.com/

[5] Digia. Qt Help, http://doc.qt.digia.com/4.6-snapshot/qthelp-framework.html

[6] Note that when generating a new help file (qch), this file must not be opened on the Assistant, otherwise you will get a permission error! To avoid this, close the assistant whenever you re-generate the help. When you open it again, you don't need to remove and add the qch again, since it will automatically update the contents to their latest version.

things, we went to the pub afterwards to chat about the talk and other stuff. I had just read Michael Feathers' excellent book *Working Effectively with Legacy Code* and so was keen to canvas opinion. I started talking to Steve Love (who had then just taken over as editor of C Vu) and it suddenly

what the conference did was to magnify the effects of those branch meetings and lunches ten fold

became apparent that although I understood the mechanics of programming and the technology what I was failing to appreciate was the philosophy of it. In essence I was taking what I was reading far too literally and failing to take the time to understand many of the underlying principles. In retrospect I suspect that this is largely the result of being a self-taught programmer that was arrogant enough to 'know it all' by the time they reached University. You don't need a type-system when you're programming in Assembler, and anything other than Assembler is 'obviously' going to be too slow…

And then I went to my first ACCU Conference in 2008. Before going I had definitely dropped a notch or two on the scale, but what the conference did was to magnify the effects of those branch meetings and lunches ten fold. The team I was currently working in were certainly as capable a bunch of programmers as any, but they weren't all the passionate kind. Suddenly I found myself surrounded by like-minded individuals and quite a few authors of the books on my desk. Now we're talking scary, but oh-so awesome too. Is it any wonder I didn't go to bed for the next 4 nights for fear of missing out on another mind-blowing revelation?

By the end of my first conference, I was definitely down to a 3 or even 2. No, what I realised is that The Programmer Scale is not linear – it's exponential! There is just so much more stuff to learn. I really hope that nobody actually asks me to rate myself in an interview because I'm not sure 2 is the answer they're looking for and they probably don't know what the far end of the scale looks like either.

What anyone who's ever read one of my reviews of the ACCU conference will have hopefully picked up on, is that everyone there is so approachable and eager to share. Yes, we all have to earn a crust, but it doesn't feel like you're in competition with anyone else, instead it feels like everyone there wants to advance The Cause together, where that cause is to be the best programmer we can.

In the intervening years I have got over my fear of asking what I previously would have thought of as a stupid question, and in fact have learnt to embrace my naivety. Yes, there is a lot of new cool technology to learn, but there are also the basics that we use every day and yet might not fully appreciate. Hearing some 'respected' individuals arguing at 4 o'clock in the morning about the definition of 'equality' goes a long way to boosting one's confidence so that you don't feel like you're the only one who thinks the topic is a little bit trickier than the books make out.

Ah, yes, confidence. Some people seem to have it in spades, almost to the point where a little humility wouldnt go amiss now and then. At the other end of the spectrum are those of us that need constant validation and reassurance that we're heading in the right direction, or that we have a worthy contribution to make. The agile practitioners tell us that face-to-face communication is worth so much more than a requirements document, and so it is with learning. Reading books might help you learn mechanics, but I've found the essence only really sinks in when discussing those concepts with others.

As I write this my first proper article is about to be published in *Overload*. I started it well over 3 years ago, and even though I was pretty confident of the subject matter, I still felt uneasy about putting it forward. Instead I decided to take smaller steps, firstly by starting a blog and then by succumbing to 'light pressure' from the C Vu editor and doing the write-ups for the ACCU London meets and annual conference. At the conference a couple of years ago I dipped my toe in the water and did a 5 min lightning talk which seemed to go down well. The following year I found myself hosting an ACCU London meet for 60 mins, and then doing a 90 min

I'm in no doubt that joining the ACCU has had a profound effect on both my career and my personality

session at the conference a few months later. This year I'm presenting again and have started moving ideas from the 'possible blog post list' to the 'potential ACCU article list'.

After being a professional programmer for almost 20 years you'd have thought things would have settled down by now. And yet I've probably learnt more about programming in the last 10 years than during the first 10. I've certainly learnt more about myself, and look forward to learning even more about both.

I'm in no doubt that joining the ACCU has had a profound effect on both my career and my personality. That has only been possible because certain people put their time and effort into ensuring that the organisation continues to function and hopefully flourish. Sadly I only get to thank some of those in person once a year at the conference. Luckily though a few others I get to meet a little more regularly and two of those in particular have been instrumental in providing the reassurance I needed to get off my backside and start to explore my own potential. Perhaps it's because they are the editors of the two ACCU publications that it gives them a licence to cajole, coax and entice us into making a contribution. But you know what, that's just what some of us need and for that, Fran & Steve, I thank you. ■

Write for us!

C Vu and Overload rely on article contributions from members. That's you! Without articles there are no magazines. We need articles at all levels of software development experience; you don't have to write about rocket science or brain surgery.

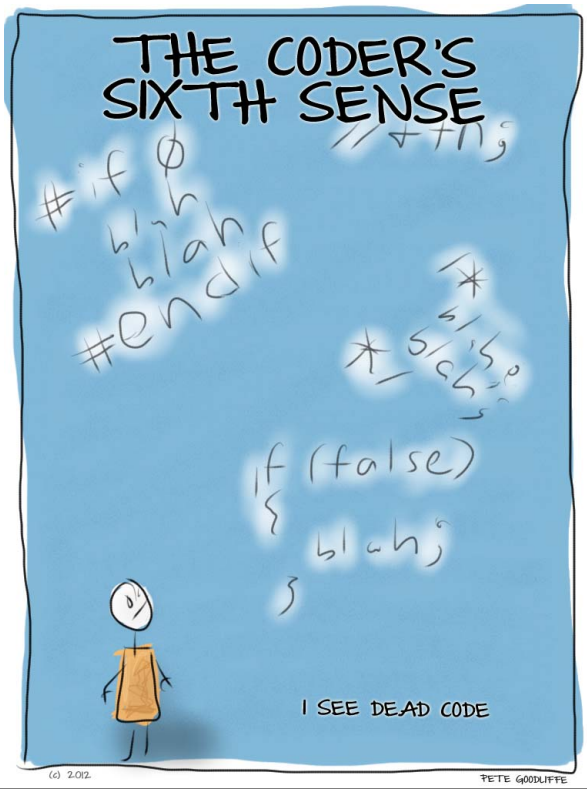
What do you have to contribute?

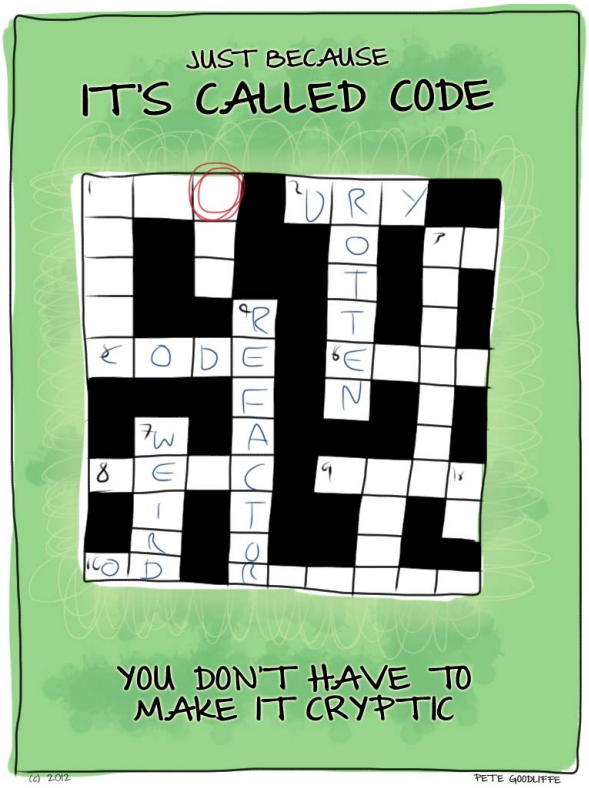
- What are you doing right now?
- What technology are you using?
- What did you just explain to someone?
- What techniques and idioms are you using?

For further information, contact the editors: cvu@accu.org or overload@accu.org

The Art of Software Development

Pete Goodliffe illustrates development practices.



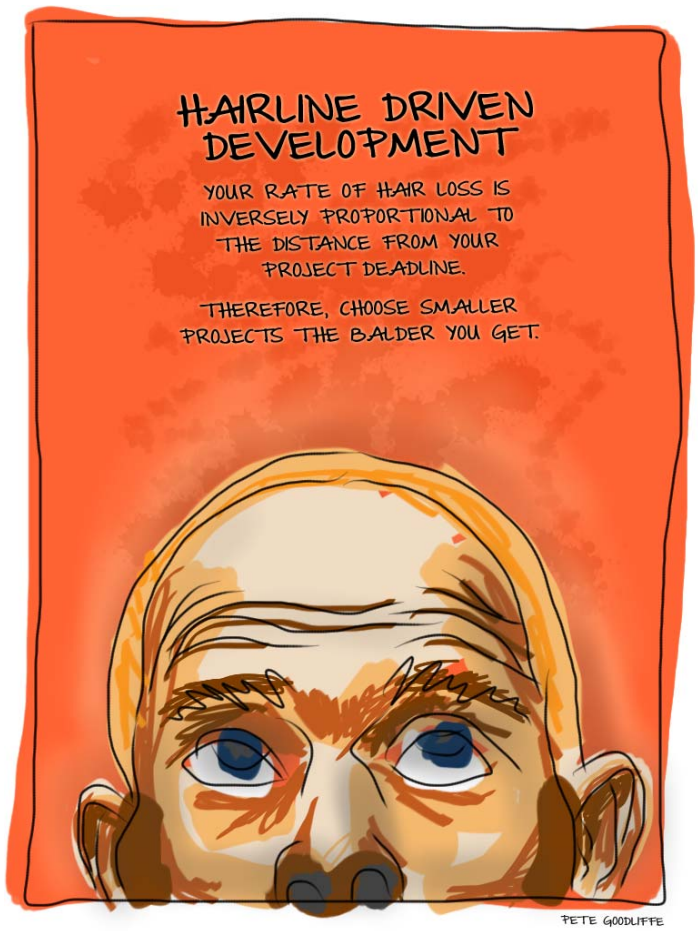


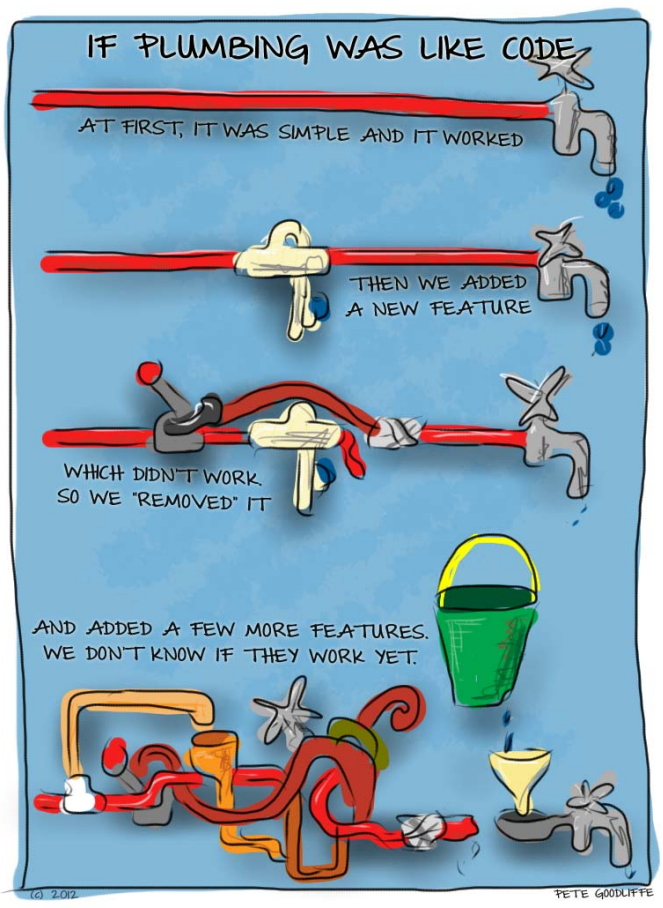
PETE GOODLIFFE

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 or @petegoodliffe


DESIGN PRINCIPLE #43:
DECOUPLE EVERYTHING
WAA-AA-AA-AA
THAT SHOULD BE DECOUPLED
(c) 2012
PETE GOODLIFFE


TIME IS NOT A GREAT HEALER
EXAMPLE #1:
PICKING UP SOME OLD SOURCE CODE
SOFTWARE DEVELOPMENT
WHO ON EARTH WROTE THIS *&!@&!! CODE?
OH. YEAH. IT WAS... ME.
(c) 2012
PETE GOODLIFFE


HAIRLINE DRIVEN DEVELOPMENT
YOUR RATE OF HAIR LOSS IS INVERSELY PROPORTIONAL TO THE DISTANCE FROM YOUR PROJECT DEADLINE.
THEREFORE, CHOOSE SMALLER PROJECTS THE BALDER YOU GET.
PETE GOODLIFFE


IF PLUMBING WAS LIKE CODE
AT FIRST, IT WAS SIMPLE AND IT WORKED
THEN WE ADDED A NEW FEATURE
WHICH DIDN'T WORK. SO WE "REMOVED" IT
AND ADDED A FEW MORE FEATURES. WE DON'T KNOW IF THEY WORK YET.
(c) 2012
PETE GOODLIFFE

In the Tool Box – Introduction

Chris Oldwood introduces a column about his weapons of choice.

For many programmers their bread and butter involves slaving over code written in a general purpose language, such as C++, C# or Java. These all require a text editor for modifying the program source code and a compiler for turning that into something the machine can actually use. Those working in the realm of the dynamic languages have a similar toolset, albeit with an interpreter or JIT compiler to turn their masterpiece into boring old machine code.

And for some this is where it ends too. The testing, deployment, documentation and project management fairies all take over now to finish off those other menial tasks. After all, the hard work's been done, right?

As the software systems we are trying to create get more and more complex, so does the number and type of tools we need to use to get the job done. In some big corporations there are teams dedicated to testing and deployment, but that still leaves an awful lot that we need to cover ourselves. In the smaller shops you've got all that along with painting your office once in a while too.

Getting from 'problem discovered' to 'fix deployed' could well see the use of a tool chest that would make an electrician feel under-equipped

We should always strive to use 'the right tool for the job', but that doesn't mean that we have to buy an all-singing, all-dancing bug tracking system if our development practices ensure that our bug count remains low – a spreadsheet may well be 'the right tool'. At least, it might for now. Later, if the spreadsheet becomes unwieldy (hopefully due to a high demand for more useful features, not bugs) we can investigate alternatives then.

The problems we need to solve outside of our core code-writing activity are often varied. After the text editor (nay, IDE) and compiler (which is often chosen for us) the Version Control System probably comes next in the list as we need to manage our code-lines as we checkout, commit, branch, merge and spelunk on a regular basis. From there it's a short hop to the Continuous Integration server that automates a set of build scripts and ad-hoc tools that even Heath Robinson would be proud of. Deployment is probably a similar affair. Even if 'all' you turn out is an .MSI something has to produce that, and you'll need to track your builds and symbols somewhere just in case a bug shows up in the field.

Oh, yes, and there is testing too. Unit testing forms the basis of many teams' testing strategy, with integration and system scale testing thrown in for good measure. Once you reach the outer layers of the system there are a plethora of tools to help with those really gnarly problems, like testing UI's and web services. Although you will probably strive to create automated tests where possible, there is still a place for manual testing and that must involve some sort of tooling too, even though you might not think of it that way.

If your product is a system composed of many moving parts, such as for batch processing or a service, you've also got the support and monitoring sides to consider. The distinction between an 'operator' and 'developer' are so blurred now that you might have to do a spot of log trawling and live debugging followed by the creation of a patch that needs to be rolled out post-haste. Getting from 'problem discovered' to 'fix deployed' could well see the use of a tool chest that would make an electrician feel under-equipped. If it's really serious there might be some major 'glue' required to hold the system together until the correct action can be implemented, tested and rolled-out.

And that's what this column wants to be about – how people solve the problems that allow us to do the other thing we actually think we're being paid to do. Any discussion of text editors is clearly out of scope on the basis that it's only mildly less contentious than discussing tabs and spaces. Similarly it's not an excuse to rope in 3rd parties to write some 'Advertorials' either. It's a chance for jobbing programmers to open up their own tool boxes and let the C Vu readership know what kinds of cool stuff they have in there and how they use it.

The good thing about metaphorical tool boxes is that they are TARDIS like in nature and so can contain everything from the svelte and simple TR through to the powerful and mind-bending GIT. Yes, we can all read the manual on how TR and GIT might be used, but what's more important is how do you use it? Why do you use it? What are some of the trade-offs that caused you to choose, say, the monkey wrench over the spanner? The aim is to try and convert some of that Unconscious Incompetence into Conscious Incompetence [1], or better yet into Conscious Competence by exposing new tools, or just reminding some of us about the old ones that are collecting dust because we've forgotten about them. Better yet how have you managed to make up for the shortfall in one tool's feature set by using another?

Clearly the classic UNIX commands such as SED and AWK provide a huge source of inspiration, but I'm also hoping we can look further afield to the less obvious. For instance, have you ever treated .csv files and Excel spreadsheets as database tables to aid in testing? Or, how do you generate a unique build number? What about the kinds of information you put on a dashboard for your support team?

Sadly the one-way communication of these pages will likely only whet your appetite and so you'll need somewhere else to go to delve into the finer points of how, what, why, when and where. For that there is always 'accu-general' – the virtual coffee shop come mailing list that never actually serves coffee, but is frequented by craftsmen of the programming trade who are always all too eager to share their opinions.

So, pop open that tool box have a rummage around and see what interesting stuff you can find. ■

References

[1] http://chrisoldwood.blogspot.co.uk/2010/04/turning-unconscious-incompetence-to.html

CHRIS OLDWOOD

Chris started as a bedroom coder in the 80s, writing assember on 8-bit micros. Now it's C++ and C# on WIndows. He is the commentator for the Godmanchester Gala Day Duck Race and can be reached at gort@cix.co.uk or @chrisoldwood

If you read something in C Vu that you particularly enjoyed, you disagreed with or that has just made you think, why not put pen to paper (or finger to keyboard) and tell us about it?

Impact of Semantic Association on Information Recall Performance

Derek Jones concludes his analysis from the latest ACCU Conference experiment.

This is the second of a two part article describing an experiment carried out during the 2012 ACCU conference; the first part was published in the last issue of C Vu.[8] This second part discusses the results from the linear relationship question that subjects answered. The experiment is derived from and takes account of results from previous ACCU conference experiments; in particular it closely replicates one performed in the 2004 ACCU experiment.[5]

To recap the description of the experiment given in the first article: subjects first saw two, unnested, **if** statements and were asked to remember the names of the variables and operators appearing in the control expressions. This information had to be recalled after they had analysed a nested **if**-statement having the following form (the results of the remember/recall question were covered in part 1):

```
if ((e > a) && (u < a))
   if (u > e)
      ............
   else
      ............
```

Subjects were asked to indicate which arm of the nested **if**-statement they thought would be executed, should the conditional expression in the first **if**-statement be true; some questions did not have a unique answer, i.e., the first conditional expression did not sufficiently constrain the values of the two variables in the second conditional expression that it was possible to unconditionally deduce whether the expression was true/false.

The analysis of subject responses to these questions is the subject of this article.

The relative order of the three variables was randomly chosen for each problem presented (the same identifiers **a**, **e** and **u**, were always used).

The hypothesis

Studies that have investigated some of the kinds of relational reasoning that people encounter in everyday life have found patterns in subjects' performance, e.g., the accuracy of answers has depended on how the original relationships were specified (see below). The hypothesis tested by this part of the ACCU 2012 experiment is that the accuracy of subjects' (i.e., developers) answers is consistent with the two patterns outlined by De Soto, London, and Handel [2] described below (also see Table 1).

Relational reasoning

The psychology of deduction uses the terms linear syllogisms or linear reasoning to describe deduction between statements involving relational operators. The term usually used to describe a (sub)expression containing a relational operator, in programming language specifications, is a relational expression.

Linear syllogisms are part of mathematical logic and the skills associated with making deductions based on relational information are usually assumed to be one of the higher cognitive abilities that humans possess. However, studies have found that a number of animals have the ability to adapt their behavior to a given situation based on relational knowledge they have previously acquired. For instance, aggressive behavior between two animals is sometimes used to determine which one is dominant, relative to the other; aggression can lead to fighting and injury and is best avoided if possible. The ability to make use of relative dominance information (perhaps obtained when watching the interaction between other members of a social group) may reduce the need for aggressive behavior during an encounter between two members of the same group who have not yet established their relative dominance through a face to face encounter (i.e., the member most likely to lose is able to deduce this outcome and behave in a subservient fashion).

A study of Pinyon Jays (a social species of birds) and Scrub Jays (a non-social species) by Pazymino[11] found that individual birds from the social species appeared to make use of relational information to work out their relative dominance while birds from the non-social species did not.

Relational reasoning in humans

If some animal brains don't possess what are considered higher level cognitive reasoning abilities and yet possess a cognitive mechanism capable of combining and making use of relational information, it is possible that humans also possess a similar mechanism (this is not to say that they don't have any other high level cognitive systems capable of performing the same task). A possible consequence of having such a special purpose, lower level, reasoning mechanism is that it may not handle all relational expressions in the same way (i.e., it is likely to be optimized for handling those situations that commonly occur in its owner's everyday life). Some of the studies of human linear reasoning have found that subjects are slower and make more errors when the operands in a sequence of relational expressions occur in certain orders.

A study by De Soto [2] used a task based on what is known as social reasoning (using the relations better and worse). Subjects were shown two premises, involving the names of three people, and a possible conclusion (e.g., Is Mantle worse than Moskowitz?) and given 10 seconds to answer "yes", "no", or "don't know".

Based on the results (see Table 1) the researchers made two observations, which they called paralogical principles (cases 5 and 6 possess both, while cases 7 and 8 possess neither):

1. People process orderings more accurately in one direction compared to others. Subjects gave more correct answers when the ordering direction was better-to-worse (case 1) than mixed direction (case 2, 3), and were least correct in the direction worse-to-better (case 4). This suggests that use of the word better should be preferred over worse (the British National Corpus [9] lists better as appearing 143 times per million words, while worse appears under 10 times per million words and it is not listed in the top 124,000 most used words).
2. People end-anchor orderings; that is, they focus on the two extremes of the ordering. In this study people gave more correct answers when the premises stated an end term (better or worse) followed by the middle term, than a middle term followed by an end term.

DEREK JONES

Derek used to write compilers that translated what people wrote. These days he analyses code to try to work out what they intended to write. Derek can be contacted at derek@knosof.co.uk

Table 1

Eight sets of premises describing the same relative ordering between A, B, and C in different ways (people's names were used in the study), followed by the percentage of subjects giving the correct answer. Adapted from De Soto, London, and Handel.[2]

	Premises	Percentage Correct Responses		Premises	Percentage Correct Responses
1	A is better than B B is better than C	60.5	**5**	A is better than B C is worse than B	61.8
2	B is better than C A is better than B	52.8	**6**	C is worse than B A is better than B	57.0
3	B is worse than A C is worse than B	50.0	**7**	B is worse than A B is better than C	41.5
4	C is worse than B B is worse than A	42.5	**8**	B is better than C B is worse than A	38.3

A related experiment in the same study used the relations to-the-left and to-the-right, and above and below. The above/below results were very similar to those for better/worse. The left-right results showed that subjects performed better with a left-to-right ordering than a right-to-left ordering.

The strategy used to solve a given problem has been found to vary between people. A study by Sternberg and Weil [15] found a significant interaction between a subject's aptitude (as measured by verbal and spatial ability tests) and the strategy they used to solve linear reasoning problems. However, a person having high spatial ability, for instance, does not necessarily use a spatial strategy. A study by Roberts, Gilmore, and Wood [14] asked subjects to solve what appeared to be a spatial problem (requiring the use of a very inefficient spatial strategy to solve). Subjects with high spatial ability used non-spatial strategies, while those with low spatial ability used a spatial strategy. The conclusion made was that those with high spatial ability were able to see that the spatial strategy was inefficient and to select an alternative strategy, while those with less spatial ability were unable to perform this evaluation.

If the evaluation of relational expressions in source code is performed using a cognitive mechanism that has been optimized for certain kinds of frequently occurring, everyday, activities then it is possible that developer performance will be good for relational expressions that match the form of these everyday activities and not so good on relational expressions that don't match. The if-statement conditional expressions used in this study permuted over all possible combinations of operator/operand ordering.

The list of questions for each subject was generated by randomising the eight possible operator/operand orderings, creating questions using this ordering, randomizing the orderings again and repeating until all of the required questions had been generated. This process was repeated when generating the problem sheets for each subject.

Threats to validity

As well as the possible threats to validity listed in part 1, the following are specific to the subject of part 2.

Although subjects were told: "Treat the paper as if it were a screen, i.e., it cannot be written on.", there was nothing to prevent them using the paper on which the questions were written as a temporary work area. Several subjects did write notes on the paper next to a few `if`-statement problems.

For those questions whose answer was that either arm might be executed some subjects wrote a question mark (i.e., ?) as their answer and some left the answer blank. Both forms of answer were treated as specifying that either arm of the nested `if`-statement could be executed. It is not possible to check whether this assumption was the intended answer.

Measurements of C source [6] show that the binary less-than operator (i.e., `<`) occurs twice as frequently as the greater-than operator (i.e., `>`), compared to the better/worse English words used by De Soto et al which has a frequency ratio of 14. It is possible that the much lower frequency ratio for the relational operators will cause the performance for both of them to be very similar.

Subjects can approach the demands of answering the problems this study in a number of ways, including the following:

- seeing it as a challenge to accurately remember/recall the conditional expression information and be willing to trade-off performance on the relational operand question;
- recognizing that would refer back is always an option, but that it is more important to correctly answer the relational operand question;
- making no conscious decision about how to approach the answering of problems.

Results

A total of 432 nested `if`-statement problems were answered by 22 subjects, of which 47 (10.9%, in 2004 the percentage was 4.7%) were incorrect. The mean number of answers per subject was 19.6 (sd 7.7), slightly lower than the 2004 mean of 21.

Subjects had a mean of 15.1 years (sd 10.2) experience writing software professionally.

The number of incorrect answers is very weakly correlated with the number of problems answered (Pearson correlation coefficient 0.24, 95% confidence interval -0.20 to 0.60). While performance on reasoning tasks has been found to decrease with age [3], subject software development experience (which is likely to be highly correlated with age) is not correlated with percentage of incorrect answers (Pearson correlation coefficient 0.02, 95% confidence interval -0.42 to 0.45) and only very weakly to the number of answers given (0.29, 95% confidence interval -0.16 to 0.64)

The error rates reported by other studies (where subjects read a problem typed on a card) were: De Soto et al [2] 39.2–61.7% (subjects were required to answer within 10 seconds rather than in their own time), Clark [1] 6%, Potts [12] 5%, Mayer [10] 4–36%, Quinton et al [13] not given, Sternberg et al [15] 1.7–3.5%. A study where subjects heard a tape recoding of the problem [4] reported an error rate of 8–19%.

In the following discussion H denotes high, M denotes middle, and L denotes low. So H > M denotes "high greater than middle" and M > L "middle greater than low". *unk* is used to denote the case where the conditional expression does not uniquely specify the relationship between all three variables.

All of the data and R code used in the analysis is available on the experiments web page. [7]

Reasoning performance

Table 2 lists the number of correct and incorrect answers for various combinations of relational operators in the outer `if`-statement, ordered by percentage of incorrect answers (the percentages from the 2004 ACCU experiment are in the last column).

Table 2

For all subjects, the total number of correct/incorrect answers and the percentage of incorrect answers for the combination of relational expressions appearing in the first two columns. The last column is the percentage incorrect in the 2004 ACCU experiment.

Left condition	Right condition	Correct	Incorrect	Percent	2004 %
M < H	L < M	51	3	5.6	5.9
L < M	M < H	33	2	5.7	3.8
H > M	L < M	40	3	7.0	5.5
M > L	M < H	40	4	9.1	5.6
M < H	M > L	39	4	9.3	4.4
L < M	H > M	34	4	10.5	2.8
M > L	H > M	42	5	10.6	6.6
H > M	M > L	37	6	14.0	1.7
unk	unk	69	16	18.8	NA

If subject behavior in 2012, for this question, was consistent with that in 2004 the relative order of percentage incorrect answers for the two years would be strongly correlated, however a Kendal rank correlation test shows a weak negative correlation (i.e., -0.29).

The following compares the results against those predicted by the hypothesis proposed in the introduction:

1. Use of the more common operator reduces incorrect answers: looking at the operands appearing in questions having the lowest and highest percentage of incorrect answers we see that these closely match the predictions made by the hypothesis (see the first two columns of Table 3), what is the probability of this occurring through a random process?

   There are 8! ways of arranging the 8 available combinations (the *unk* case is ignored here); there are two ways in which the two less-than operators can occur first, one way a greater-than can occur last and 5! different ways of ordering the other possibilities, giving a probability of

   $$\frac{2!5!}{8!} \Rightarrow 0.006$$

   for this combination occurring at random (well below a p-value of 0.05).

   While the 2012 behavior matches this hypothesis the results from the 2004 experiment do not; in fact for 2004 the lowest incorrect percentage combination and second highest percentage are swapped, almost the opposite of the proposed hypothesis.

2. End-anchoring: The operand ordering that is the complete opposite of this end-anchoring pattern has the lowest percentage of incorrect answers, orderings that follows this pattern have the second lowest percentage and highest percentage of incorrect answers. The Middle value appears as the first operand twice (for both left and right relational expressions) in the lowest incorrect percentage and the high incorrect percentage; there is no evidence of any end-anchoring.

Figure 1 shows the percentage of incorrect answers given by each subject, ordered by increasing percentage. Just under half of the subjects do not give any incorrect answers; perhaps the analysis will reach a different conclusion if subjects who give very few incorrect answers are excluded. Table 3 only includes results from subjects whose answers were at least 6% incorrect. There is only one change in the relative ordering, M>L H>M moves up to 4th from 7th.

Table 3

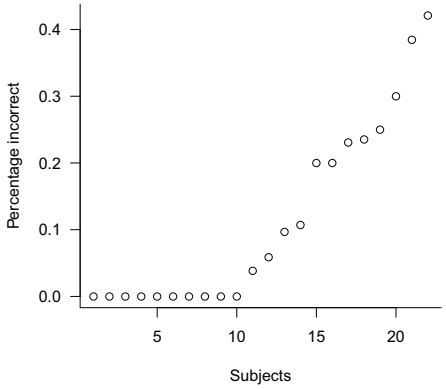
A subset of Table 2 created by only including results from those subjects whose percentage of incorrect answers was greater than 6%.

Left condition	Right condition	Correct	Incorrect	Percent
M < H	L < M	22	3	12.0
L < M	M < H	10	2	16.7
H > M	L < M	15	3	16.7
M > L	H > M	19	4	17.4
M > L	M < H	17	4	19.0
L < M	H > M	12	3	20.0
M < H	M > L	13	4	23.5
H > M	M > L	14	6	30.0
unk	unk	32	16	33.3

In those cases where the conditional expression in the outer if-statement did not contain enough information to uniquely specify which arm of the nested `if`-statement would execute, the first arm was incorrectly given in 10 answers and the second arm in 6 answers. If we assume there is an equal probability of either arm being incorrectly specified, then there is a 12% chance of 10 out of 16 incorrect answers specifying one particular arm.

Figure 1

Subjects listed in order of increasing percentage of incorrect answers.



There were 9 answers specifying that either arm was possible but in fact the correct answer to the question was one particular arm (5 for one arm, 4 for the other).

Interaction between remember/recall and reasoning questions

Having to answer the first part of the problem (i.e., remembering information about the variables in the control expression of two if-statements) ties up cognitive resources (e.g., short term memory decays over time and unless regularly refreshed it is soon lost), leaving less resources to process the nested `if`-statement problem.

Figure 2 shows that the percentage of incorrect answers in the relational question is not correlated with percentage of correct answers in the operand remember/recall question.

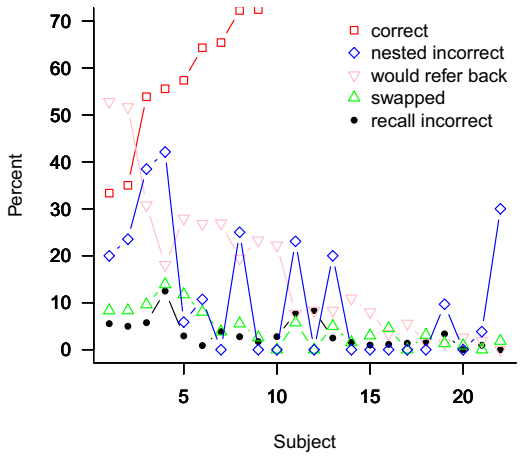
However, there is quite a good correlation between incorrect answers and percentage of swapped operand answers to the remember/recall question (Pearson correlation coefficient 0.66, with a 95% confidence interval of 0.34 to 0.85, p-value = 0.00074); the correlation with percentage of incorrect operand answers is not quite so good (0.55, with a 95% confidence interval of 0.17 to 0.79, p-value = 0.0076).

Discussion

A surprisingly high percentage (45%) of subjects gave no incorrect answers. This observation was not noted in 2004 because the low number of incorrect answers given by subjects meant that zero incorrect was not surprising (the 2004 figure was 44% subjects giving no incorrect answer).

Figure 2

The percentage of operand recall correct, relational incorrect, would refer back, swapped and operand recall incorrect answers for each subject. Subjects are ordered by percentage of operand recall correct answers given (scale clipped to expand relational view).



Looking at Figure 2 many of the subjects who had a very low percentage of incorrect answers also had a very low percentage of incorrect remember/recall answers. There appears to be a group of subjects whose performance on the two questions in this experiment was much better than the other subjects.

There is a good correlation between incorrect answers to the nested **`if`**-statement question and percentage of swapped operand answers in the remember/recall question. Both of these findings are consistent with subjects' having problems processing the relative ordering of identifiers seen in code.

There are two notable differences in subject performance between 2004 and 2012:

- the percentage of incorrect answers is more than twice as high in 2012; the figure is reduced from 10.9% to 7.2% if *unk* answers are excluded;
- there is no correlation between the form of conditional expression and the relative incorrect answer rate in the results from the two years.

and two differences in the questions answered:

- the remember/recall question involved assignment statements in 2004 and the operands of **`if`**-statement conditional expressions in 2012;
- in 2004 the nested **`if`**-statement relational expression question always had an answer that was one of the two arms, while in 2012 the question could have the answer that either arm might be executed.

Were the differences in the questions used the main contributing factor in the differences in subject responses seen in the two years?

More experiments are needed to find out why nearly half of the subjects gave no incorrect answers (or alternatively why just over half gave incorrect answers) and to find out what caused subject performance to vary on almost the same question (in 2004 and 2012).

Conclusion

There were two groups of subjects who exhibited their own consistent behavior in answering the two questions in this experiment:

- Approximately 40% of subjects gave a very low percentage of incorrect answers to both questions (i.e., zero or one incorrect answer);
- approximately 25% of subjects showed some tendency to mix up the order of operands appearing in both questions. ■

Further reading

For a readable introduction to human reasoning see *Reasoning and Thinking* by Ken Manktelow. *The Cognitive Animal* edited by M. Bekoff, C. Allen, and G. M. Burghardt contains 57 short, wide ranging, essays (of varying quality) on animal cognition.

Acknowledgments

The author wishes to thank everybody who volunteered their time to take part in the experiment and the ACCU for making a conference slot available in which to run it.

References

[1] H. H. Clark. Linguistic processes in deductive reasoning. *Psychological Review*, 76(4):387–404, 1969.

[2] C. B. De Soto, M. London, and S. Handel. Social reasoning and spatial paralogic. *Journal of Personality and Social Psychology*, 2(4):513–521, 1965.

[3] A. S. Gilinsky and B. B. Judd. Working memory and bias in reasoning across the life span. *Psychology and Ageing*, 9(3):356–371, 1994.

[4] J. Huttenlocher. Constructing spatial images: A strategy in reasoning. *Psychological Review*, 75(6):550–560, 1968.

[5] D. M. Jones. Experimental data and scripts for short sequence of assignment statements study. http://www.knosof.co.uk/cbook/accu04.html, 2004.

[6] D. M. Jones. The new C Standard: An economic and cultural commentary. Knowledge Software, Ltd, 2005.

[7] D. M. Jones. Experimental data and scripts for impact of semantic association on information recall performance. http://www.knosof.co.uk/dev_experiment/accu12.html, 2012.

[8] D. M. Jones. Impact of semantic association on information recall performance. *C Vu*, 24(6):3–8, Jan. 2013.

[9] G. Leech, P. Rayson, and A. Wilson. *Word Frequencies in Written and Spoken English*. Pearson Education, 2001.

[10] R. E. Mayer. Qualitatively different encoding strategies for linear reasoning premises: Evidence for single association and distance theories. *Journal of Experimental Psychology: Human Learning and Memory*, 5(1):1–10, 1979.

[11] G. Paz-y-Miño C, A. B. Bond, A. C. Kamil, and R. P. Balda. Pinyon jays use transitive inference to predict social dominance. *Nature*, 430:778–781, Aug. 2004.

[12] G. R. Potts. Storing and retrieving information about ordering relationships. *Journal of Experimental Psychology*, 103(3):431–439, 1974.

[13] G. Quinton and B. J. Fellows. 'Perceptual' strategies in the solving of three-term series problems. *British Journal of Psychology*, 66:69–78, 1975.

[14] M. J. Roberts, D. J. Gilmore, and D. J. Wood. Individual differences and strategy selection in reasoning. *British Journal of Psychology*, 88:473–492, 1997.

[15] R. J. Sternberg and E. M. Weil. An aptitude strategy interaction in linear syllogistic reasoning. *Journal of Educational Psychology*, 72(2):226–239, 1980.

Comparing Algorithms

Richard Polton takes different approaches to a well-known problem.

This article is going to demonstrate a number of different approaches to solving Problem #1 from the Project Euler problem set. The construction of each of these approaches will be discussed and, as a side-effect, the performance of each of these implementations of Problem #1 will be observed. Some possible reasons for the difference in performance between each of the implementations will be presented and discussed.

Spoiler Alert! This code contained within this article will enable you to calculate the solution to a problem from Project Euler. Specifically we have tackled problem #1 (http://projecteuler.net/problem=1) here. If you have not already solved this problem and you wish to do so, go and do it now then come back here before continuing reading this article. Thanks.

Problem statement

Problem #1 is very simple, as you might expect given its position in the list of the Project Euler problems. It is stated as follows:

- If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.
- Find the sum of all the multiples of 3 or 5 below 1000.

See, simple! As it is so simple it means that we can explore some programming techniques without losing ourselves in the complexity of the problem statement (only in the solution implementation).

Method

The first thing, fairly obviously, is to determine the actual answer to the posed problem. We're not going to give that away directly but rest assured that the following really does solve the problem correctly.

Given the problem description, it will be clear that the calculation will not be time-consuming and will almost certainly be completed in a very small amount of time. In order to generate useful results we decide to observe the length of time taken to perform the calculation 25000 times in succession. This test will then be performed multiply (say, 15 times) and the average length of time will be determined. We do this for each of the supplied implementations.

To most people who have grown up on a steady diet of C-related languages, imperative programming comes naturally and so that is probably the route that most would take to solve this problem. We will present a number of imperative solutions as well as some functional solutions.

The code examples which are presented in this article have been written using F#, a functional [2] programming language in the .NET ecosystem. Recognising that not all readers will have written F# themselves, a quick overview of a couple of language features follows. Firstly, data are immutable unless specifically indicated otherwise. Therefore, if we wish to generate a sum of elements by adding subsequent values to a 'running' total then we need a mutable field, indicated by the **`mutable`** keyword. Mutable fields in F# have special syntax for updating their values, namely the **`<-`** operator. Secondly, almost everything in F# is a function which returns a value (the exception in the example code being the **`for`** statement).

Therefore, **`if`** is a function which must return a value from each of its branches. This means that it is necessary to return a value from both the **`then`** and the **`else`** branch, whereas were we writing using an imperative programming language we would almost certainly have ignored the **`else`** branch because it would be considered a no-op.

Listing 1

```
let mutable sum = 0
for i = 1 to upperLimit-1 do
  sum <-
    if (i%3 = 0 || i%5 = 0)
    then sum + i
    else sum
```

Listing 2

```
let mutable sum = 0
for i = upperLimit-1 downto 1 do
  sum <-
    if (i%3 = 0 || i%5 = 0)
    then sum + i
    else sum
```

We offer fifteen different implementations of the problem. The first four present variations on an iterative imperative loop, the next six present implementations using higher-order functions, the next pair present recursive solutions and the final three present alternative implementations of an optimised algorithm.

Let's look at them in sequence. Listings 1 and 2 all present slight variations on the iterative theme. Listing 1 is a natural encoding of the problem in its simplest form which is then wrapped in a loop so that our framework can repeatedly perform the calculation. That is, we create a mutable placeholder for the sum and then we loop over the range under consideration analysing integer each in turn. If the integer satisfies our predicate it is added to the sum.

Listing 2 is a reverse iteration. That is, although the same calculation is performed as in Listing 1, it is done in the reverse order. Therefore, this function starts from the upper limit and adds progressively smaller numbers to the 'running' total.

Alternate implementations of Listings 1 and 2 (available on blogspot [1]) extract the mutable placeholder initialisation from inside the loop that repeats the tests. This was done to demonstrate any possible performance cost from recreating the mutable field each time.

Listings 3 to 8 all make use of the higher-order functions. Listing 3 and Listing 4 use the **`List`** higher-order functions while Listing 5 uses the **`Seq`** higher-order functions. This builds the list, 'pipes' it into a filter where the list is reduced and then folds the remaining integers into a sum.

Listing 6, Listing 7 and Listing 8 reproduce Listing 3, Listing 4 and Listing 5 respectively but by iterating in the reverse direction, ie downwards from the upper limit. Listing 4 and Listing 7 both construct the range of integers outside of the loop. These are the equivalent of the alternate implementations of Listings 1 and 2 previously mentioned, and exist for the same purpose. Note that it does not make sense to construct the **`seq{}`** outside of the solution loop because **`seq{}`** is a lazily-evaluated range.

RICHARD POLTON

Richard has enjoyed functional programming ever since discovering SICP and feels heartened that programming languages are evolving back to LISP. He likes 'making it better' and enjoys riding his bike when he can't. He can be contacted at richard.polton@shaftesbury.me

Listing 3
```
let sum =
  [1..upperLimit-1]
    |> List.filter (fun i -> i%3=0 || i%5=0)
    |> List.fold (fun state i -> i+state) 0
```

Listing 4
```
let sum =
  input
    |> List.filter (fun i -> i%3=0 || i%5=0)
    |> List.fold (fun state i -> i+state) 0
```

Listing 5
```
let sum =
  seq {1 .. upperLimit-1}
    |> Seq.filter (fun i -> i%3=0 || i%5=0)
    |> Seq.fold (fun state i -> i+state) 0
```

Therefore, each individual element in the sequence is generated as it is used.

At last we have two recursive solutions. Listng 9 is a simple recursive implementation. The termination condition is triggered when the counter reaches the upper limit of the range and the base-case of the recursive function is the binary decision, as it was before in the iterative implementation, between those integers which are multiples of 3 or 5 and those which are not. Note that this function is not tail-recursive and so makes relatively heavy use of the stack as each recursive call creates a stack frame.

The second recursive solution in Listing 10 uses the ACCUMULATOR pattern. This has the effect of reversing the order in which the elements are summed. This function is tail-recursive and therefore all the stack frames between the initial recursive function call and the last can be elided by the OS / VM / compiler which we would expect to result in a quicker and less memory-intensive implementation.

Finally, we present an optimised algorithm implemented in three ways: iteration, recursion and recursion using an accumulator. This algorithm benefits from the observation that it is not necessary to consider every possible candidate integer, only those which are multiples of three or five. This has the effect of reducing the number of divisions necessary as we only need to check that a multiple of three is not also a multiple of five to prevent double-counting. Listing 11 is an iterative implementation while Listing 12 and Listing 13 are the recursive and recursive with accumulator implementations respectively. The 'trick', if you will, in the recursive implementations is to realise that both counters need to be increased in each function call but not every value will be required. This issue still occurs in the iterative solution but can be performed in a separate step.

Each of the implementation snippets above is wrapped in an appropriately-named function, as in Listing 14 (which is the function wrapping the code from Listing 1). Given definitions of **`maxLoops`** (the number of times we wish to repeat the calculation in a single timed run), **`upperLimit`** (one

Listing 6
```
let sum =
  [upperLimit-1 .. -1 .. 1]
    |> List.filter (fun i -> i%3=0 || i%5=0)
    |> List.fold (fun state i -> i+state) 0
```

Listing 7
```
let sum =
  input
    |> List.filter (fun i -> i%3=0 || i%5=0)
    |> List.fold (fun state i -> i+state) 0
```

Listing 8
```
let sum =
  seq {upperLimit-1 .. -1 .. 1}
    |> Seq.filter (fun i -> i%3=0 || i%5=0)
    |> Seq.fold (fun state i -> i+state) 0
```

Listing 9
```
let rec Impl3p counter acc =
  if counter=upperLimit
  then acc
  else
    if counter%3=0 || counter%5=0
    then Impl3p (counter+1) (acc+counter)
    else Impl3p (counter+1) acc
let sum = Impl3p 1 0
```

Listing 10
```
let rec Impl3p counter =
  if counter=upperLimit
  then 0
  else
    if counter%3=0 || counter%5=0
    then counter + Impl3p (counter+1)
    else Impl3p (counter+1)
let sum = Impl3p 1
```

Listing 11
```
let mutable sum = 0
let mutable loopDiv3,loopDiv5 = 0,0

// loopDiv3 increases slower than loopDiv5
while loopDiv3<upperLimit do
  loopDiv3 <- loopDiv3 + 3
  loopDiv5 <- loopDiv5 + 5

  // if we haven't already added it in loopDiv5
  sum <-
    sum + (if loopDiv3 < upperLimit
      then (if loopDiv3 % 5 <> 0
        then loopDiv3
        else 0)
      else 0)
  sum <-
    sum + (if loopDiv5 < upperLimit
      then loopDiv5
      else 0)
```

Listing 12
```
let rec Impl4p counterDiv3 counterDiv5 =
  if counterDiv3>=upperLimit
    then 0
    else (if counterDiv3 % 5 <> 0
      then counterDiv3
      else 0) +
        (if counterDiv5 < upperLimit
        then counterDiv5
        else 0)
    + Impl4p (counterDiv3+3) (counterDiv5+5)

let sum = Impl4p 0 0
```

Listing 13
```
let rec Impl4p counterDiv3 counterDiv5 acc =
  if counterDiv3>=upperLimit
  then acc
  else Impl4p (counterDiv3+3) (counterDiv5+5) (
    (if counterDiv3 % 5 <> 0
    then counterDiv3
    else 0) +
      (if counterDiv5 < upperLimit
      then counterDiv5
      else 0) + acc)

let sum = Impl4p 0 0 0
```

Listing 14

```
let Impl1a = fun () ->
  let whichImpl = "Iteration"
  st.Restart()
  for loop = 1 to maxLoops do
    let mutable sum = 0
    for i = 1 to upperLimit-1 do
      sum <-
        if (i%3 = 0 || i%5 = 0)
        then sum + i
        else sum
  st.Stop()
  st.ElapsedMilliseconds, whichImpl
```

greater than the maximum value in the range of integers under consideration) and **st** (a .NET StopWatch object).

Each of the implementation snippets above is wrapped ina n appropriately named function, which returns the elapsed time and the English name of the function, as in Listing 14.

In order to run these independent pieces of code automatically, a small harness function was created. It takes the function to run as its parameter, **whichImpl**, and it runs the name of the implementation and the average execution duration. **res** is, therefore, the 2-tuple containing the elapsed time in milliseconds and the name of the specific method being tested. **name** contains the English name from **res** and **elapsed** contains the list of elapsed times from **res**. The function then iterates through the list of elapsed times printing them to the console in turn (using a procedure, ewww!). Finally we calculate the mean elapsed time by folding **elapsed** and dividing it by the number of elements and return **name** and the average, as shown in Listing 15.

Listing 15

```
let run (whichImpl : unit -> int64 * string) =
let res =
  [ for count = 1 to howMany do
    yield whichImpl() ]
  let name = res |> List.head |> second
  let elapsed = res |> List.map first
  elapsed |> List.iter (fun elem ->
    printfn "%s : %i iterations took %i ms"
      name maxLoops elem)
  let avg = float (elapsed
    |> List.fold (fun state i -> state+i) 0L)
    / float (List.length elapsed)
  name, avg
```

Results

The results measured in milliseconds when run using F# Interactive on my testing PC are given in Figure 1.

Conclusion

We have shown a number of algorithmic approaches to solving this simple problem and have quantified them each in terms of average speed of execution. While we cannot draw any hard conclusions from the observed times, and if we ignore the optimised algorithms, we can see that reverse iteration and recursion with an accumulator were approximately equivalent and were the quickest on the test PC (using F# Interactive) and that using the **Seq** higher-order functions were an order of magnitude slower.

From the observations of the various implementations using higher-order functions it seems reasonable to infer that it is the construction of the integer range which leads to this massive increase in elapsed time. This merits further investigation in an explicit Release build.

For the full, runnable code please head over to blogspot [1] and scrape the F#. Note that F# uses indentation to delimit nested code, eg loop bodies, and so you will almost certainly need to re-ident. Sorry about that. ■

Notes and references

[1] This is the blogspot post: http://randomgemsandmiraculousdiscoveries.blogspot.co.uk/2013/01/which-is-quicker-iteration-or-recursion.html

[2] Not strictly true. F# also supports imperative programming constructs, such as 'for', higher-order iteration functions like List.iter and mutable data, each of which are demonstrated herein. F# additionally supports OO constructs such as classes and interfaces, although we show no examples of these in this article.

Further reading

These links might be of interest.

- The first is from 'How to Design Programs' and is the section relating to the creation of recursive functions using the ACCUMULATOR pattern: http://htdp.org/2003-09-26/Book/curriculum-Z-H-39.html
- The second is a discussion on Stack Overflow relating to the performance differences observed and expected between simple recursion and accumulator-based recursion in the Racket Scheme dialect: http://stackoverflow.com/questions/4733456/performance-of-recursion-vs-accumulator-style
- The final link is to a Wikipedia article discussing Tail Calls in general (and tail-recursion as part of that): http://en.wikipedia.org/wiki/Tail_call

Figure 1

```
[("Iteration", 252.8);
 ("Iteration with mutable creation outside loop", 255.0666667);
 ("Reverse iteration", 225.0);
 ("Reverse iteration with mutable creation outside loop", 236.1333333);
 ("Iteration using List higher-order fns", 3509.933333);
 ("Iteration using List higher-order fns with pre-compiled integer range", 386.6);
 ("Iteration using Seq higher-order fns", 4848.933333);
 ("Reverse Iteration using List higher-order fns", 3679.466667);
 ("Reverse Iteration using List higher-order fns with pre-compiled integer range", 396.7333333);
 ("Reverse Iteration using Seq higher-order fns", 4883.266667);
 ("Recursion", 242.4);
 ("Recursion with Accumulator", 231.9333333);
 ("Iteration incrementing a pair of counters", 48.6);
 ("Recursion using a pair of counters", 90.4);
 ("Recursion with Accumulator using a pair of counters", 53.26666667)]
```

Code Critique Competition 80

Set and collated by Roger Orr. A book prize is awarded for the best entry.

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.

Last issue's code

Thanks to Francis Glassborow for sending me this critique. If you see code that could be in a code critique, please send it in!

I keep getting a compiler warning when I compile the code with gcc. Please help me get rid of it!

```
gcc cc79.c -o cc79
cc79.c: In function 'addFirst':
cc79.c:36:17: warning: assignment from
incompatible pointer type [enabled by default]
cc79.c: In function 'printList':
cc79.c:46:9: warning: assignment from
incompatible pointer type [enabled by default].
```

I've tried with Microsoft VC and that gives me an even more confusing error:

```
cc79.c(36) : warning C4133: '=' : incompatible
types - from 'LLNODE *' to 'LLNODE *'"
```

The code is in Listing 1.

Listing 1

```
/* This program builds a basic linked list. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct { // Define a linked list node
  char *name;
  struct LLNODE *next;
}LLNODE;

void addFirst(char *data); //Declare function
                           //prototypes
void printList(void);

LLNODE *head = NULL; //Define global pointer
                     //variable head

int main(int argc, char *argv[])
{
  addFirst("Peter");
  addFirst("Paul");
  addFirst("Mary");
  printList();
  return EXIT_SUCCESS;
}
void addFirst(char *data)
{
  LLNODE *newNode;
  newNode = malloc(sizeof(LLNODE));
  newNode->name = data;
  newNode->next = head; //errors here
  head = newNode;
}
void printList(void)
{
  LLNODE *itr = head;
  while(itr != NULL)
  {
    printf("%s\n", itr->name);
    itr = itr->next;  //And errors here.
  }
}
```

Critiques

Henrik Austad <henrik@austad.us>

Looking at the error message:

assignment from incompatible pointer type [enabled by default]

gives the first hint: gcc doesn't really know what this is, in other words, **struct LLNODE** is not something gcc 'knows' about.

```
typedef struct { // Define a linked list node
  char *name;
  struct LLNODE *next;
}LLNODE;
```

Not going into the issue of (mis)using **typedef** for hiding values, the **struct** is anonymous which makes it hard to reference from inside the **struct** itself. By changing to

```
typedef struct llnode {
  char *name;
  struct llnode *next;
  ...
} LLNODE;
```

then the warning goes away and gcc is able to determine the type of ***next**.

/me sits back and wait for the inevitable flame of "don't you know compilers? This particular instance is called <something I should've remembered>" :)

Amar Sanakal <amar@sanakal.com>

The errors talk about incompatible pointer types and we see that both the LHS & RHS are of type **LLNODE**. Now this should make us suspect how **LLNODE** is defined.

We see that it is defined as:

```
typedef struct { // Define a linked list node
  char *name;
  struct LLNODE *next;
} LLNODE;
```

At first glance everything looks okay, until we take a closer look at how **next** is defined. We have 2 problems with the **struct** definition:

1. The **struct** itself does not have a name. This by itself is not a problem.

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

2. Which leads us to the next problem in defining **next**, where it would be ideal to just say **LLNODE *next;** but we can't say that as the **typedef** is not yet available at that point.

There are two ways of fixing this. One is to separate out the **struct** definition and the **typedef** as follows:

```
struct llnode { // Define a linked list node
  char *name;
  struct llnode *next;
};
typedef struct llnode LLNODE;
```

The other way, if you prefer to still keep them together, give the **struct** a name and use that to define the **next** member as follows:

```
typedef struct llnode {
  char *name;
  struct llnode *next;
} LLNODE;
```

Using either of these approaches resolves the issue. Attached [Ed: but not included in this critique] is the updated source that has both the above resolutions conditionally compiled using a preprocessor directive. If you compile it using the command:

```
gcc -Wall -o cc79 -save-temps -DSTRUCT cc79.c
```

you can see the 1st approach used in the generated `cc79.i` file. Similarly, compiling using the command:

```
gcc -Wall -o cc79 -save-temps cc79.c
```

you can see that the second approach has been used in the `cc79.i` file.

Some additional notes about the compilation:

- it is always a good practice to use **-Wall** option to capture all kinds of warnings.
- we used the **-save-temps** option to save all temporary files created during compilation, in particular the **.i** file which contains the output after preprocessing has completed which we wanted to check.
- we use (or not) the **-DSTRUCT** option to make the preprocessor choose the different paths of the **#ifdef** directive.

PS. I have tested/checked this only on gcc, but I suspect that MSVC should behave similarly.

Jon Kalb <jon@kalbweb.com>

The warning that we are being given about 'incompatible pointer type' is a clue that the next data member isn't being defined as we expect. The member is defined as a pointer to **struct LLNODE**, but there is no such beast. There is an anonymous **struct** that has been t**ypedef**'d to **LLNODE**, but there is no **struct LLNODE**.

These leads to the question, if **next** is a pointer to something that isn't defined, why does the compiler not report an error on the line: **struct LLNODE *next;**? Since the definition of **next** is as a pointer, the compiler doesn't need to know any details about **struct LLNODE** so it treats this as if it were a forward declaration and accepts **struct LLNODE *** as a valid type, expecting **struct LLNODE** to be defined later (before details of its definition are needed).

The fact that this is self-referential is not the issue. You could also include a data member **foo** defined thusly:

```
typedef struct {
   char *name;
   struct LLNODE *next;
   struct BAR *foo;
} LLNODE;
```

This is also a legal definition of **LLNODE** and the compiler will not emit an error (nor likely a warning) until we try to use **foo** without first defining **struct BAR**. This definition of **struct BAR** must be a definition of a **struct** named **BAR**, not a **typedef** named **BAR**. This is not sufficient:

```
typedef struct {
  int bar;
} BAR;
```

It must be of this form:

```
struct BAR{
  int bar;
};
```

or:

```
typedef struct BAR{
  int bar;
} BAR;
```

Otherwise **struct BAR** is undefined and using **foo** will result in a warning or error, depending how it is being used.

This code also has issue in failure cases. If **malloc()** returns **NULL**, which it might, the dereferences of **newNode** on lines 35 and 36 will not end well. The definition of **printList()** passes **itr->name** to **printf()** without verifying that it is non-nil. This would succeed by printing **"(null)"** on many, but not all implementations. With the usage of **addFirst()** as written here, this isn't an issue, but if **addFirst()** were called with **NULL**, a subsequent call to **printList** may not end well. A rewrite of **addFirst()** could address both of these issues.

```
// Returns NULL on node allocation failure.
LLNODE *addFirst(char *data)
{
  LLNODE *newNode;
  newNode = malloc(sizeof(LLNODE));
  if (newNode) {
    // Use an empty string when passed NULL.
    static char empty = 0;
    newNode->name = data? data: &empty;
    newNode->next = head;
    head = newNode;
  }
  // so callers can detect failure.
  return newNode;
}
```

Balog Pál <pasa@lib.hu>

Let's start with the immediate problem. We get a warning when using member **next**. We defined it as **struct LLNODE ***. So to use it with a matching type, we need a **struct LLNODE** around. But the code has no such thing.

We did define a type with name **LLNODE**, that is an alias to a **struct**. The submitter probably thought that that is **struct LLNODE**, but without a name it is just that: a **struct <unnamed>**. And pointers of two unrelated **struct**s are indeed incompatible as pointed out in the warning.

A follow-up question could be that why then no complaints about using the undefined **struct**? It's because we used it only where an incomplete type is good enough. The code only uses it to define **next**. But that pointer is never dereferenced in the code. If we tried say **itr->next->next**, that would not compile.

Another follow-up question could ask why no clash on name **LLNODE** if we succeeded in making it have multiple meanings. Well, that goes back to early C design, where the names of structure tags go in a separate name table, and must be referred with the **struct** prefix, rather than in the 'global' name table. As a matter of fact that is the usual motivation to use **typedef**s for **struct**s. **typedef** puts its name in the regular table, so no prefixing is needed. And referring to the original **struct** is rarely needed, so it can be left unnamed. The case in our example where we need a pointer to this **struct** as a member is rare.

So how to fix the warning? The simplest way with the existing code is just to make **next void***. We'll have all the same conversions, but C rules for **void*** allow it in both directions without a notice. Certainly just like the warning goes away on benign cases it will go away on bad cases too in the future, say if we mistakenly assign to **->next** instead of **->name**.

The solution that fixes the case preserving most type safety is to give the `struct` a name and use that. Here we have multiple options. One is to name the `struct tagLLNODE` and use that defining `next`, leaving everything else. We can remove the `typedef`, just stating `struct LLNODE`, and use `struct` prefix on every use. Or we can combine those using the name `LLNODE` for the `struct` and leave the `typedef` with the same name. In practice the middle option is rarely used, and choice between the other two is made by the local style guide that is made up on coin flip or religion.

Now to the general issues. My first and most important advice is to not do it at all. Start with switching to C++, and use `list`, `vector` or other fine collections in the standard lib. There's pretty little excuse to write stuff in C in 2013, like 'no C++ compiler for the desired target', but the preamble stated access to both gcc and MSVC. Even if you're positive to stay with C, get a framework or a library with these most basic constructs. Linked list support is so fundamental it is in each one of them. And for learning purposes it's better to read the established good material, starting with the usage description before the code. I mean it.

Let's look at the code. The node structure usually has the `next` pointer at the front, and the content after it. For this little sample it's indifferent, but in life we can have many lists and more content in this node, uniform look would make us shuffle it later. Why not start right that? The content now is a `char*`, that may or may not be good, but to tell we should know the intent of this list.

Then we have prototypes with odd comment. Side comments shall apply to exactly one item, the one it stands beside. And shall provide information. That we see a prototype or a variable does not qualify. Though if we move it in the front of the previous line, it could make a banner comment, creating a 'section' in code, which is quite usual. We can see a correct prototype of the second function with (void) rather than `just ()`, maybe due to the gcc warning. ;-) And `addFirst` takes a `char*` argument: that is a yellow flag. Oddly the functions do not have an argument for the list to act on. Also normally I'd expect a `printList` function be told where to print, but it may just be part of testing.

Then we see a global variable that is a red flag in its own right. It might be okay if just used for a test case, but we start to suspect that the function will work on this thing as a list, that would be kind of a showstopper on my review.

In main `addFirst` is called with string literals. It does compile – even in C++ – but is generally a bad idea. Though kept for backward compatibility in the language we should never use that conversion and bind string literals only to `const char*`. As `char*` would make it too easy to modify the content which leads to undefined behavior.

In function `addFirst` (that I'd rather call `addFront`) we see that a new node is allocated from the heap, and put correctly in the chain. A local variable is created at front of block and assigned later, that is not very good, even in C that picked up the define-with-initialization-anywhere syntax with C99. But at the very front of block even the oldest C compiler allows to make the variable and initialize it immediately. Even better, as we have no intent to change the value, `newNode` shall be `const`.

When we see `malloc()`, we shall see two other things. One, is a matching `free()` somewhere. The other is the check of the return value, as failure to allocate the requested memory is reported by returning `NULL`. And if that happens the next statement dereferences it for undefined behavior. So it shall be checked, and something done about it. Not easy to tell what. And indeed it's PITA to make every `malloc` call surrounded with checks, traps, deallocation, etc. But that is C, and if one did not heed my advice to drop it, one must bite the bullet. The lack of `free()` call anywhere means that we leak the memory. Fortunately it's not that hard to handle, just provide a `freeList` function that iterates the list and process all nodes.

`printList` looks correct though it were better written using the `for()` instead of `while`. After all we have both the init and the iteration part.

Besides the coding details we have a general design issue. The current one takes a pointer to the content data and places it directly in the list. That fact must be documented for the function, as it puts the obligation on the caller to keep the pointer valid. It's all too easy to pass a pointer to a local variable, that gets out of scope and out of life while still pointed to in the list. For this reason collections often make a copy of the data making it stable; going this route our inserter function would take `const char *`, and call `strdup`. This mitigates concerns about string literals and allows safe modification of the content in the list through `char *`. Releasing the memory is not significant hassle, as it's just an extra line beside the one releasing the node.

But if the intent was just to collect literals, avoiding the copy is fair game, just add the `const` and proper documentation.

I already mentioned that the functions handling the list should take the list head as parameter and act on that rather than on a global. IMO implementing lists in general makes little sense instead of using stock ones, but implementing a list handler that can be used on just a single object is questionable.

Pete Disdale <pete@papadelta.co.uk>

The 'presenting problem' is straightforward to fix, and caused by a confusion between `struct`s and `typedef`s. In the original definition

```
typedef struct {
  char *name;
  struct LLNODE *next;
} LLNODE;
```

what is `LLNODE`? Is it a `struct` or a `typedef`? It is plainly a `typedef`, so the expression `struct LLNODE` is wrong; the `next` member must reference a `struct` but a self-referential anonymous `struct` such as this one cannot be referenced as it is, well, anonymous. Splitting the definition up, the above is semantically equivalent to

```
struct somename {
  char *name;
  struct somename *next;
};
typedef struct somename LLNODE;
```

So a self-referential struct must have a name or tag, and simply changing the original definition to

```
typedef struct _llnode {
  char *name;
  struct _llnode *next;
} LLNODE;
```

makes the compilation error go away and the resulting executable runnable; at least it does on the oldish (3.4.5) version of gcc I have to hand here. But of course, there are more dragons lurking and ready to unleash their wrath :-) Some are basic problems, some trivial, and some that would spoil one's day if `addfirst()` were unleashed as part of a larger system.

1. General pedantry

The code as written adds new list entries to the head of the list. One assumes that this is intentional, but does result in

```
addfirst("Peter");
addfirst("Paul");
addfirst("Mary");
printlist();
```

printing out the list in the reverse order to which the entries were added. This is probably more of an issue for the way my aged brain has been wired in that 'Mary, Paul and Peter' somehow doesn't gel...but changing the name of `addfirst()` to `addhead()` or `addstack()` might be a bit more descriptive of the FILO nature of the function.

2. Check those return values!

`malloc()` can fail, so its return code must be checked before assigning to the `NULL` that might be returned. `addfirst()` should contain at the very least

```
void addfirst (char *data)
{
  LLNODE  *newnode = malloc (sizeof(LLNODE));
  if (newnode == NULL)
```

```
  {
  fprintf (stderr, "malloc: out of memory\n");
  return;
  }
  // ... safe to assign to 'newnode'
```

And of course, for every **malloc()** there must be a corresponding call to **free()**! So after the call to **printlist()** in **main()**, there should be a similar one to **freelist()**:

```
void freelist()
{
  LLNODE  *itr = head;
  while (itr != NULL)
  {
    LLNODE *thisnode = itr;
    itr = itr->next;
    free (thisnode);
  }
}
```

The only point worth mentioning here is the sequencing: the next node to be freed must be remembered before the current one is freed. I.e. if the code were written as

```
while (itr != NULL)
{
  free(itr);
  itr = itr->next;
}
```

it is just possible that the value in **itr** is no longer valid after calling **free()** and hence **itr->next** could contain garbage. Perhaps unlikely but nonetheless possible if the library implementation of **free()** calls some internal compaction or defragmentation routines.

This issue could be avoided by using a recursive algorithm such as

```
void freelist(LLNODE *lnptr)   /* UNTESTED! */
{
  if (lnptr != NULL)
  {
    freelist (lnptr->next);
    free (lnptr);
  }
}
```

but this obviously requires a parameterized **freelist()** [see the revised code below which has this] and whilst the code is more compact it will require a large stack for a large data set!

3. Pointers and pointers-to-data

Now to the interesting bit :) The code as presented makes discrete, successive calls to **addlist()** with [static] string literals, so the assignment

```
newnode->name = data;
```

just 'works'. In Real Life however, it is far more likely that the list will be populated either from user input or reading a file, in both cases using a common buffer to initially accept the data. So for example **main()** might look more like

```
char buffer[BUFSIZ];
FILE *fp;
while (fgets(buffer, sizeof(buffer), fp)
       != NULL)
  addfirst(buffer);
```

The problem is obvious: each time that **addfirst()** is called the name member is set to the address of buffer, so **printlist()** will simply display whatever data buffer currently points to; at best the last data read in however many times and at worst garbage if buffer has been freed or reused in between times. **addlist()** should therefore store a ***copy*** of the data, for example

```
newnode->name = strdup(data);
```

And, needless to say **newnode->name** needs to be checked for **NULL** and appropriate action taken as above!

4. Scoping

Variables (and functions come to that) should be no more visible than absolutely necessary in any language that supports scoping, and global variables in C should be avoided whenever possible. In a tiny program like this one it isn't a huge deal, but tiny programs have a habit of growing as they are developed or incorporated into other programs, and global variables make the code ever harder to maintain and sooner or later come back to bite :)

In this exercise, there is no need whatsoever for **head** to be global; only **main()** needs to know about it (OK, that is a sort of global in this case, but **main()** would likely be called **create_list()** or some such if the code were incorporated into a larger body of code). Any dependant code simply needs to be passed a reference to it as and when needed, and to pass it back if modified.

So, putting all the above into practice, here is a new version of the original listing.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct _llnode
{
  char    *name;
  struct  _llnode *next;
}
LLNODE;
/* Function prototypes */
LLNODE  *addfirst(LLNODE *head,
  const char *data);
void    printlist(LLNODE *head);
void    freelist(LLNODE *head);
char    *dummy_fgets(char *buf,
  size_t bufsize);
// argc, argv are unused but left in for now
int main (int argc, char *argv[])
{
  LLNODE  *head = NULL;
  char buffer[32];
  // pretend we're reading input from a file
  while (dummy_fgets (buffer, sizeof(buffer))
         != NULL)
    head = addfirst(head, buffer);
  printlist (head);
  freelist (head);
  return 0;
}
/*
 * addfirst()
 * Add 'data' to the head of the list,
 * returning an updated pointer to its head.
 * If the malloc() fails, just return the
 * pointer to the existing head of list.
 */
LLNODE  *addfirst (LLNODE *head,
  const char *data)
{
  static char malloc_err[] =
    "malloc: out of memory\n";
  /* Note that malloc() is cast to a pointer
     to LLNODE */
  LLNODE  *newnode =
   (LLNODE *) malloc (sizeof(LLNODE));
  if (newnode == NULL)
  {
    fprintf (stderr, malloc_err);
    return head;
  }
  if ((newnode->name = strdup (data)) == NULL)
  { /* free the just allocated 'newnode'! */
    free (newnode);
```

```
    fprintf (stderr, malloc_err);
    return head;
  }
newnode->next = head;
return newnode;
}
/*
 * Print out the list starting at 'lnptr'.
 */
void    printlist (LLNODE *lnptr)
{
  while (lnptr != NULL)
  {
    printf ("%s\n", lnptr->name);
    lnptr = lnptr->next;
  }
}
/*
 * Free the list starting at 'lnptr'.
 */
void    freelist (LLNODE *lnptr)
{
  while (lnptr != NULL)
  {
    LLNODE *thisnode = lnptr;
    lnptr = lnptr->next;
    free (thisnode->name);
    free (thisnode);
  }
}
/*
 * dummy_fgets()
 * A quick and dirty function to emulate
 * reading the input data from a file as if
 * using fgets().
 * Returns the next item in the 'names' array
 * or NULL on reaching "EOF".
 */
char *dummy_fgets(char *buf, size_t bufsize)
{
  static const char *names[] =
  { "Peter", "Paul", "Mary" };
  static int i = 0;
  if (i < sizeof(names) /sizeof(*names))
  { // use strncpy to prevent buffer overrun
    strncpy(buf, names[i++], bufsize - 1);
    buf[bufsize - 1] = '\0';
    return buf;
  }
  return NULL;
}
```

5. And finally...

There is a little 'trick' I came across a few years back that simplifies the `malloc()` for such simple list nodes and which also produces slightly more efficient memory allocation. It might be already widely known but I'll include it anyway; change the `LLNODE` definition to

```
typedef struct _llnode {
  struct  _llnode *next;
  char    name[1]; /* This MUST be the final
                      struct member! */
} LLNODE;
```

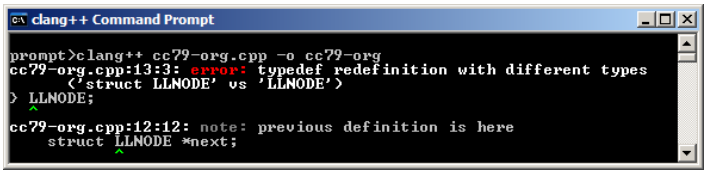
and the `addfirst()` function to

```
LLNODE  *addfirst (LLNODE *head,
  const char *data)
{
  LLNODE *newnode = (LLNODE *)
    malloc (sizeof(LLNODE) + strlen(data));
  if (newnode == NULL)
  {
    fprintf (stderr,
      "malloc: out of memory\n");
    return head;
  }
  strcpy (newnode->name, data);
  newnode->next = head;
  return newnode;
}
```

This results in a single call to `malloc()` for a buffer large enough to hold the `struct` *and* the data (the `data[1]` ensures that there is room for the null-terminator) and simplies error handling. The only thing to watch out for is that the `data` member *must* be the last `struct` member, for reasons which I hope are obvious :)

Martin Moene <m.j.moene@eld.physics.LeidenUniv.nl>

In programming, any problem can be solved by introducing another level of indirection [1]. I applied the essence of the code critique format to use someone else's head another time and fed the C-source to the clang++ compiler for advice [2, 3].



Compiling a C program as a C++ program may work out as C and C++ share a common subset: A valid C program may also be a valid C++ program [5]. Therefore an error in a C source may also be sensibly diagnosed by a C++ compiler – may.

Missing `struct` tag The compiler gives us a useful hint where to look for the error, err warning in C: the struct tag `LLNODE` referred to in line 12 is missing from `typedef struct` on line 9. Note that it is the lucky consequence of an interestingly subtle difference between how C and C++ handle the `struct` tag that is helping us: In C++, `struct` declarations act like they are implicitly `typedef`ed, as long as the name is not hidden by another declaration with the same name [6]. Due to the `struct` tag missing from line 9, the tag seems to refer to an unrelated type defined elsewhere. Had the `struct` tag name been different from the `typedef`ed `LLNODE`, the error would have surfaced much later, namely where the C warnings were at lines 36 and 46.

To satisfy both C and C++ we can write:

```
typedef struct LLNODE
{
  char *name;
  struct LLNODE *next;
}
LLNODE;
```

With this change, the program compiles as a C program without warnings. In C++ the `typedef`ed `LLNODE` hides `struct LLNODE`'s name. Further, to compile as C++, line 34 requires a C-cast (`LLNODE*`), or a `static_cast<>()`.

Another solution to get rid of the warnings is to suppress them via for example option `-w` and compile with: `gcc -w cc79.c -o cc79` (!)

Gripes The most notable issues are the use of global data such as `LLNODE *head` and the absence of error handling. The two functions that operate on the list hide that fact from their prototype. Provide the list as a parameter to these functions to improve locality and testability. In a similar vein, provide the stream to print to as a parameter to `printList()`. Memory allocation and stream output can and should must be tested for errors. Concluding positively: parameterise from above [7] and handle errors.

Design Qua design I'm missing a list abstraction that separates the creation of nodes from managing them.

Other remarks Reading the code we see that the strings that are added to the list are not copied, but only referenced by pointer. In the way the list

is used here, that's not a problem, because the strings are in scope as long as they are used. Moreover they are literal strings and have the lifetime of the program [8]. Maybe the intention was to use `strdup()`, but failing that we may as well omit inclusion of `<string.h>` (Found with Visual Lint).

With a high warning level such as `-Wall -Wextra` [-Weffc++] the compiler helps us to discover further weaknesses.

There are several locations where `const` can be used to help prevent programming errors. Visual Lint explicitly states it and g++ and clang++ give an indication for this via the warning: deprecated conversion from string constant to `char*` at the call sites of `addFirst()`. To add `const`, change the code to `char const *name;` on line 11 and void `addFirst(char const * const data)` on lines 15 and 31. Make similar changes on lines 12, 19 and 42 to (`struct`) `LLNODE const *`.

Parameters `argc` and `argv` of `main` are not used and changing the declaration of `main` to `int main(void);` prevents unused parameter warnings. In a program this small we could as well use short names such as `add()` and `print()` as it's all about the list.

'List'-ing I had a go at a conventional linked list in C, then quickly switched to a more functional approach and came up with something that looks like [9]:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef __cplusplus
# include <stdbool.h>
#endif
typedef char const * Text;
typedef struct Node
{
    bool valid;
    Text text;
    struct Node const * next;
}
Node;
// ...
int main(void)
{
  return apply( print,
                cons( "Peter",
                cons( "Paul",
                cons( "Mary", list() )))
              ).valid
    ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

The code uses in-channel, out-of-band error signalling. Further it assumes sufficient memory or the assistance of a garbage collector.

References

[1] Wikiquote. http://en.wikiquote.org/wiki/Computer_science

[2] A tool contains canned knowledge, presumably coming from (other) people's heads.

[3] Clang: Expressive diagnostics, GCC compatibility, http://clang.llvm.org/; See also [4]

[4] Comparison of Diagnostics between GCC 4.8 and Clang, http://gcc.gnu.org/wiki/ClangDiagnosticsComparison

[5] Bjarne Stroustrup's FAQ. Is C a subset of C++?, http://www.stroustrup.com/bs_faq.html#C-is-subset

[6] StackOverflow. Difference between 'struct' and 'typedef struct' in C++? http://stackoverflow.com/questions/612328/difference-between-struct-and-typedef-struct-in-c

[7] Kevlin Henney. The PfA papers. ACCU *Overload* 80, 81, 82, 83. 2007, 2008. http://accu.org/index.php/journals/1470

[8] StackOverflow. 'life-time' of string literal in C. http://stackoverflow.com/questions/9970295/life-time-of-string-literal-in-c

[9] Rough-edged code can be obtained from: http://www.eld.leidenuniv.nl/~moene/Home/papers/accu/cvu251-cc79/accu-cvu251-cc79-moene.tar.gz

Pawel Zakrzewski <pawel@zakrzewski.cc>

There are several issues with the presented code. The main one is caused by the author using an unnamed `struct` and inadvertently forward-declaring a completely unrelated `struct LLNODE` on line 8. That is why the compiler warns about the assignment operations between pointer to an unnamed `struct` and pointer to incomplete `struct LLNODE`. On the bright side, this operation is perfectly well defined by the standard, because the pointer is always converted back to the `LLNODE typedef` type.

As unnamed `struct`s are impossible to forward-declare, the fix requires either giving the `struct` a name or changing the type of next pointer to `void*`. I decided to use the former, because it is more descriptive and gives the compiler more information to help the programmer in the future.

Another thing that is wrong with the code is complete disregard for freeing allocated memory. While it won't be a problem in a program that simple, it's a bad habit to have. I added function `deleteList` to fix that.

The next issue with the proposed solution is usage of global variables. It's done on two levels: not only the head node is a global variable, but also both functions operating on the linked list are using that global variable. The head node being a global isn't necessarily wrong, but it could be a sign of poor design. Basing the functions on said variable, however, prevents them from ever being used in different context and doesn't allow to operate on multiple instances of list. I decided to pass appropriate pointers to all functions operating on `LLNODE` lists and to move the head node to scope of `main` function; if it ever needs to be global it will be trivial to move it out.

The last thing that doesn't sit well with me is the type of string pointer used. There are two possible design choices here, neither is suggested by the code. The list could own the strings – hence the string name being defined as pointer to non-`const char`, or it could just store pointers to strings without owning them, as it originally did, but it would stand to reason to define name as pointer to `const char` then. I decided to go with non-owning pointers, frankly, because it means less changes and it fits proposed usage. I changed all functions to expect `const char*` instead of `char*` to help the compiler prevent the user from causing undefined behaviour by writing to a string literal.

Overall this code made me feel awkward. Although I didn't spot any undefined behaviour, it does seem to dare the user to use it incorrectly and cause it.

[Ed: full solution code was supplied but is omitted to save space]

Commentary

I think between them the critiques cover just about all the issues with the code. This difference between a `struct` and a `typedef` is a common source of confusion, particularly in C, and I do wonder whether the compiler messages could be improved…. As a couple of people pointed out C and C++ have slightly different rules with name lookup of `struct`s and `typedef`s. For once I find the rules mean C++ code in this area is almost always simpler and clearer than the equivalent C code.

I was a little disappointed that no-one pushed back about what the success criteria was for the main program – presumably this was some sort of test for the linked list but it always returns `EXIT_SUCCESS` – does this mean any output is valid? Pete did realise the output would have the names in reverse order ('Peter, Paul and Mary' was a pop group from the 60s) but without some check in the code it is not obvious whether the writer of the program expected that or not. Asking what are the success criteria of tests is a good habit to develop even for such simple programs.

The Winner of CC 79

All entrants correctly identified and solved the original problem and many picked up other issues of leaking memory, use of global variables, etc. I

was quite surprised to find that Martin's list-like solution would compile as both C and C++ but wasn't sure I would really want to put this in the hands of a relatively novice C programmer…

Pal, Pete and Pawel all mentioned deleting the list: this is actually surprisingly tricky to get right because of the need to free the node after capturing the next pointer (so I think Pal should have provided a solution.) Pete's first solution, using the global head, doesn't reset it to **NULL** which could result in a dangling pointer.

I was very impressed with the conciseness of Pawel's critique which covered almost every issue in a short space; but eventually decided that Martin's critique was the one best deserving of the prize. I thought his repeated mention of using static analysis tools such as Lint would be useful to equip our novice to find some of the issues with their code on their own.

Code Critique 80

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

I have been starting to use IPv6 and have tried to write a routine to print abbreviated IPv6 addresses following the proposed rules in RFC 5952. It's quite hard – especially the rules for removing consecutive zeroes. Can you check it is right and is there a more elegant way to do it?

Here is a summary of the rules:

Rule 1. Suppress leading zeros in each 16bit number

Rule 2. Use the symbol "::" to replace consecutive zeroes. For example, 2001:db8:0:0:0:0:2:1 must be shortened to 2001:db8::2:1. If there is more than one sequence of zeroes shorten the longest sequence – if there are two such longest sequences shorten the first of them.

Rule 3. Use lower case hex digits.

The code is in Listing 2.

Readers struggling with IPv6 may seek consolation from Verity Stob: http://www.theregister.co.uk/2012/08/21/verity_stob_ipv6/

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

```
/* cc80.h */
#include <iosfwd>
void printIPv6(std::ostream & os,
  unsigned short const addr[8]);
/* cc80.cpp */
#include "cc80.h"
#include <iostream>
#include <sstream>

namespace
{
  // compress first sequence matching 'zeros'
  // return true if found
  bool compress(std::string & buffer,
    char const *zeros)
  {
    std::string::size_type len =
      strlen(zeros);
    std::string::size_type pos =
      buffer.find(zeros);
    if (pos != std::string::npos)
    {
      buffer.replace(pos, len, "::");
      return true;
    }
    return false;
  }
}
void printIPv6(std::ostream & os,
  unsigned short const addr[8])
{
  std::stringstream ss;
  ss << std::hex << std::nouppercase;
  for (int idx = 0; idx != 8; idx++)
  {
    if(idx) ss << ':';
    ss << addr[idx];
  }
  std::string buffer(ss.str());
  compress(buffer, "0:0:0:0:0:0:0:0") ||
  compress(buffer, "0:0:0:0:0:0:0") ||
  compress(buffer, "0:0:0:0:0:0") ||
  compress(buffer, "0:0:0:0:0") ||
  compress(buffer, "0:0:0:0") ||
  compress(buffer, "0:0:0") ||
  compress(buffer, "0:0");
  // might be spare colons either side of
  // the compressed set
  while (compress(buffer, ":::"))
    ;
  os << buffer;
}

/* testcc80.cpp */
#include <iostream>
#include <sstream>
#include "cc80.h"
struct testcase
{
  unsigned short address[8];
  char const *expected;
} testcases[] =
{
  { {0,0,0,0,0,0,0,0},
    "::" },
  { {0,0,0,0,0,0,0,1},
    "::1" },
  { {0x2001,0xdb8,0,0,0,0xff00,0x42,0x8329},
    "2001:db8::ff00:42:8329" },
};

#define MAX_CASES sizeof(testcases) /
sizeof(testcases[0])

int test(testcase const & testcase)
{
  std::stringstream ss;
  printIPv6(ss, testcase.address);
  if (ss.str() == testcase.expected)
  {
    return 0;
  }
  std::cout << "Fail: expected: "
    << testcase.expected
    << ", actual: " << ss.str() << std::endl;
  return 1;
}

int main()
{
  int failures(0);
  for (int idx = 0; idx != MAX_CASES; ++idx)
  {
    failures += test(testcases[idx]);
  }
  return failures;
}
```

Listing 2 (cont'd)

Standards Report

Mark Radford looks at some features of the next C++ Standard.

Hello and welcome to another standards report. I was going to begin by welcoming you to the first one of 2013, then I realised that was the January report. But of course, the January report was written earlier in 2012, just. This is the first one I've actually written this year.

I've now been writing these reports for six months and, when I started writing them, the forthcoming UK ISO C++ meeting (to be held in Bristol) seemed a long way off. Now the meeting is just a little more than two months away. In July last year we were short of £8,000 worth of sponsorship needed to make the event happen. Since then things have changed somewhat: specifically, the committee voted to extend the meetings to six days (they used to be five days). Naturally this has affected the costs. Happily, Google have now agreed to sponsor the event, but more sponsors are still being sought. If you can help, please get in touch with Roger Orr who is the organiser (or get in touch with me, and I'll put you in touch with Roger).

Many thanks to Chris Oldwood for giving me some feedback on these reports. Chris said he'd like to see more about what drives the standards committee's decisions, and that gave me an idea regarding what I'm going to talk about this time. Two factors that currently influence C++ development heavily are the need to add useful high level components to the library, and making the language easier to use by removing various 'irritations' that regularly need to be programmed around. Unfortunately I couldn't attend the recent BSI C++ Panel meeting, but I've been reading through the minutes and I see that proposals that were reviewed at the meeting are examples of both of these influences.

Traditionally, C++ has been a language of 'nuts, bolts and washers': that is, it has given users the low level tools with which to build the components they need. However, other languages (Java and C#, for example) are providing large libraries of ready made high level components 'out of the box' and ready to be used. This is where C++ has now been behind for some years, and the emphasis on addressing this is a big influence on the standard library development for C++17. For this reason, readers who take a look at the standards committee papers (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/) will have noticed many higher level proposals appearing. A very simple example is a `split()` library function. Why has C++ never had this function? It is simple to split a string on a delimiter in a few lines of code using a loop and the `getline()` function, but why do we need to keep writing a few lines of code to perform such a simple operation? Most other languages (in the same space) have always had the `split()` function! At least we now have a proposal for one (N3510). However, the road to standardisation is littered with potholes that even the (apparently) simplest proposal can trip over (which is why these proposals are discussed by many people)! For example, consider a couple of points raised in the BSI Panel meeting: (i) the `split()` proposal relies on the ranges proposal (N3513), so discussions on ranges need to be fairly advanced first, and (ii) how will `split()` cope with quoted strings (given that splitting CSV data is one of the most common uses for this function)?

Another factor influencing the language's evolution is ease of use: there are several examples of 'irritations' that have to be programmed around. Another way of looking at this is: there are several examples, in C++, where the programmer must program with (what you might call) a lack of elegance. Programming languages that support elegant coding styles lead to code that is more easily understood and, therefore, less error prone. It follows that the role of support for elegance should not be underestimated. Two such proposals that were discussed (and that received support) at the BSI Panel meeting are: Operator Bool for Ranges (N3509), and Compile Time Integer Sequences (N3493). Obviously, if you want to go into the details of these proposals, you can read the papers.I'll just give a flavour of what they're about.

Consider code like this:

```
if (DerivedType* derived =
  dynamic_cast<DerivedType*>(base))
{
  ... etc ...
}
```

This works because of the implicit test for the pointer being null. Compare with this:

```
std::string s = f();
if (!s.empty())
{
  ... etc ...
}
```

This is what Operator Bool for Ranges addresses, by advocating the addition of an explicit operator `bool()` to all strings, containers and ranges. This would allow coders to write code like this:

```
if (std::string s = f())
  use(s);
```

And like this:

```
while (std::string s = f())
  use(s);
```

Personally I think code like this looks simple and pleasantly intuitive, and it gives code the kind of elegance I was referring to.

Another factor influencing the language's evolution is ease of use

The advent of variadic templates meant that tuples could be added to the standard library in C++11. Now, we are seeing proposals that build on them. One example is Mike Spertus' proposal (N3404) that I talked about in my last report (*CVu* 24.6). Another is Jonathan Wakely's Compile-Time Integer Sequences (N3493). In Python, as Jonathan observes, tuple expansion can be done automatically: `func(*t)`. Note that in Python tuples are first class language features. In C++, with tuples in the library, it's not so easy. For example in:

```
template<class... T> void foo(std::tuple<T...> t);
```

performing an operation on each element of `t` would require extra code because `t` is not a parameter pack. Jonathan goes on to propose library solutions that support a function object being applied to the elements of a tuple.

That wraps up this standards report. Next time I'm hoping to cover the forthcoming April ISO C++ meeting taking place in Bristol. I say 'hoping' because, unfortunately, these meetings and *CVu* submission deadlines are slightly out of phase, so we'll have to see.

MARK RADFORD

Mark Radford has been developing software for twenty-five years, and has been a member of the BSI C++ Panel for fourteen of them. His interests are mainly in C++, C# and Python. He can be contacted at mark@twonine.co.uk

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-glamorous 'not recommended' rating, you are entitled to another book completely free.

Thanks to Pearson and Computer Bookshop for their continued support in providing us with books.

Jez Higgins (jez@jezuk.co.uk)

The Well-Grounded Java Developer

By Benjamin J. Evans and Martijn Verburg, published by Manning, ISBN 978-1617290060

Reviewed by Neil Youngman

The Well-Grounded Java Developer is not simply a book about Java. It's aim is to improve your Java development by not only introducing you to the latest features of Java, but also a range of development techniques and other languages that use the Java Virtual Machine.

Part 1 of The Well-Grounded Java Developer (Chapters 1 and 2) introduces the new features in Java 7.

Chapter One introduces the set of small changes that originated in project Coin. Although these are small changes, they include some features that will bring significant benefits in simplifying code, such as allowing Strings in `switch` statements and 'try with resources'. Chapter One also defines the distinction between the Java language and Java platform and summarises the issues that have to be considered when the language or platform change. One notable omission is that it does not state whether the libraries are considered to be part of the language or the platform. This chapter is written in a clear and easy to follow style.

Chapter 2 covers the new file and Network I/O facilities introduced in Java 7. This chapter tries to cram a lot of material into a small chapter, which results in a lot of examples with sketchy explanations. This chapter is best read with a Java 7 reference available to fill in the gaps. While I would have appreciated more detail, this is an introduction, not a reference and there is enough information to give a clear picture of the new APIs.

East Anglia MongoDB User Group

Paul Grenyer reviews the inaugral local meeting.

When I saw the advert on the 10gen [1] site for sponsored regional meetup groups and decided to start one I thought I'd be lucky to get 10 people join the group and half that number to the meetups. Yet again, I was wrong! Despite being an advocate of the technical community in Norwich, it still takes me by surprise just how vibrant it is. As I write this the membership of the East Anglia MongoDB User Group [2] stands at 46 and we had between 25 and 30 people come to the first meeting on Wednesday. We even had some of the regular SyncNorwich [3] crowd from Ipswich!

I wanted to start with the big guns, so tonight we had Ross Lawley [4] from MongoDB [5] creators 10gen come and give us an overview [6] of MongoDB and its features and configuration. Although the presentation was very high level, it gave just the right level of detail. At about 35 minutes it was just the right length and led to my second pleasent surprise! There was at least 20 minutes of questions from the group following the presentation. I was really pleased to get this level of interest from the group and even more pleased that everyone was able to get an indepth answer from Ross.

The Reindeer [7] was a great venue too. Several people arrived early to eat and stayed behind afterwards. The small room they have at the back of the pub was just the right size for 30 people theatre style. Although it would be a struggle if we got any bigger.

Next time we have Trisha Gee [8], also from 10gen speaking to us on Wednesday 10th April. Signup at http://www.meetup.com/EastAnglia-MongoDB/events/95196662/.

PAUL GRENYER

Paul Grenyer is a husband, father, software consultant, author, testing and agile evangelist. He can be contacted at paul.grenyer@gmail.com

References

[1] http://www.10gen.com/
[2] http://www.meetup.com/EastAnglia-MongoDB/
[3] http://www.syncnorwich.com/
[4] http://uk.linkedin.com/in/rosslawley
[5] http://www.mongodb.org/
[6] http://paulgrenyer.blogspot.co.uk/2013/01/east-anglia-mongodb-user-group-first.html#
[7] http://www.thereindeerpub.co.uk/
[8[http://uk.linkedin.com/in/trishagee

ACCU Information

Membership news and committee reports

View from the Chair

Alan Griffiths
chair@accu.org

One of the challenges of writing this column is that it appears in print over a month from the time that I write it. That means that things can change. As I write a discussion is starting regarding a draft for a revised constitution; as you read this there will be a motion to be considered and voted on that proposes changing the constitution. There may even be several motions addressing different aspects of the change. This will be the first time that members who don't attend the AGM have a chance to vote – don't waste the opportunity to have your say.

From comments I've seen elsewhere or received in private communication it seems that many members do not realise how committee meetings are currently held. Changes have been happening over the last couple of years and increasingly committee members have attended meetings remotely.

Strange though it may seem, a bunch of techies like us do not have a long list of technologies that work for remotely attending committee meetings. There are a few commercial ones that one or more committee members have seen work in a controlled environment, but for the ad-hoc variety of kit used by committee members the most successful technology is Google hangouts. While this does require attendees to have a Google Plus account (which I can imagine some don't want) it works and the price (free) is no barrier to participation.

Bookcase (continued)

Part 2 covers vital techniques that a 'well-grounded' developer needs to understand, from dependency injection to performance tuning, via concurrent programming and class loading.

Chapter 3 covers dependency injection. While the basic concept is clearly explained, I found the detail lacked sufficient context to understand how it all fits together. Sub-sections on qualifiers used by the injector and those used by the target do not state where each qualifier is used and some listings contained parts from both sides. While it is possible to work this out, it makes for a harder read than necessary and made me feel uncertain. Ultimately I had to go and read the web introduction to Guice before I really felt I understood the bulk of the material in this chapter.

Chapter 4 covers concurrency in Java. This starts with a brief primer on concurrent programming, then takes you through the older concurrency classes and introduces the newer structures that provide simplified support.

Chapter 5 covers topics related to the JVM, such as class loading, byte code and some new features of the Java language and the JVM. Although I have been using Java for some time, this is an area that has been a bit of a black box for me and this chapter was very useful in improving my understanding of the JVM.

Chapter 6 is about performance tuning. This is always a difficult topic and this chapter provides a useful mix of pragmatic advice and technical detail. I have not had a need to tune my Java apps, but when I do, I am sure this chapter will provide a good starting point.

Part 3 is about polyglot programming. As well as Java there a large number of other languages implemented on the JVM. Chapter 7 talks about the limitations of Java and its relative suitability for various kinds of project. Chapters 8, 9 and 10 introduce three of the major alternatives: Groovy, Scala and Clojure. Groovy is the most similar to Java, Scala provides a more functional programming style than Java and Clojure is a lisp variant on the JVM. All these languages are also interoperable with Java. They can call Java code and be called from Java where needed.

Polyglot programming is mixing and matching the various languages available on the Java Virtual Machine to improve productivity by applying their different strengths appropriately.

Part 4 is mainly about applying development best practice to polyglot programming projects. This covers test driven development (ch 11), build and integration (ch 12) and web frameworks (ch13). As you would expect, it covers tools in more than one language and discusses their integration with Java.

Finally, chapter 14 looks to the future with an exploration of the changes expected in Java 8, many of which will assist in supporting languages other than Java, both by adding support for new features in the JVM and by technical improvements in memory allocation and concurrency support.

Overall this book covers a wide sweep of technologies and techniques in the Java ecosystem. Given the breadth of its scope it can not be an in-depth guide to any of them, but it is very informative and well written. This book will open many people's eyes to a range of new and valuable ideas. I recommend it to any Java developers who wish to widen their horizons.

Windows System Programming 4th Edition

By Johnson M. Hart, published by Addison-Wesley, ISBN 978-0321657749

Reviewed by Stefa Turalski

It is a bit peculiar to pick-up the 4th edition of Johnson M. Hart's *Windows System Programming* – especially now, 3 years after it was published in February 2010 – a book introducing improvements in Windows 7 and Windows Server 2008 with (a bit too) elaborate C code samples whilst there is shiny new Windows 8 and Windows RT with which Microsoft steps away from desktop into tablet market.

To be honest, I don't recommend this book if you aren't a developer deeply interested in inner workings of the Windows platform, who somehow hasn't read one of previous editions and isn't busy moonlighting on an iOS or Android pet projects. Thus, if Windows is your chosen (or forced) platform, it's the book for you (at least until an updated edition arrives).

With the growing popularity of web frameworks, software hosted in the cloud or mobile system keeping us away from hardware and internals of the OS, it seems that there is little to worry about. However, even though we rarely write low level system code these days, it's still relevant to know how things work under the bonnet. Even if hidden below various layers. these basic components provided by the OS are the building blocks of every system. *Windows System Programming* starts coverage of these building blocks with details of Windows File System, Unicode, character processing, system registry, and moves on to exception handling, memory management, memory mapped files, process management, threads and synchronisation primitives (covering advanced concepts like NT6 condition variables), to cover the IPC, sockets programming, skimming over DLL entry points and thread-safety, Windows Services, asynchronous I/O and finishes with security discussions, topped off with couple of substantial appendixes. The author didn't skip a subject worth mentioning!

In fact the book belongs to an endangered species – a very well structured, solid and thought-through textbook, which, surprise, leaves the reader with additional exercises at the end of every chapter. Be assured that you will find concise discussion followed by almost self-explanatory code samples covering most of the subjects touched.

To the advantage of *Windows System Programming*, and in distinction to another great book on the subject, Richter's *Windows via C/C++* (also recently updated), most of the OS concepts are discussed with a note on how things are done in UNIX world.

I think you get the point, it's just one of these must-read book for a Windows developer.

At the last committee meeting I was asked to mention here an issue we discussed regarding the 'hardship fund'. This was originally created to support the memberships of individuals who could not finance themselves (one scenario I remember was people living in countries where it was impractical to pay in sterling). However, there have been decreasing calls on this fund over the years and while it is still possible to donate, nothing has been paid out for quite some time. The committee needs guidance on how to proceed: Should we continue accepting contributions to the fund? And what do we do with the money already donated?

We do sometimes offer concessionary memberships to members in financial difficulties. So one option for using the hardship fund could be to 'make up' the difference (effectively transferring the money to our general budget). It would also be possible to spend the money in new ways (supporting attendance at the Conference for example). Perhaps you can suggest something better?

As I've already indicated,, when you read this you will be a month closer to the AGM than I am while writing it. And so this will be the last 'From The Chair' you read before the AGM (but not the last I write).

Over the past year I've tried to organise the committee so that things can be done, but this still requires people to step up and do them. Nowhere is this more clear than with the ACCU website. The committee stands ready to support making progress in updating it but no-one is currently making any progress on the larger parts of the problem. It is a big problem and without someone willing to manage the work I don't think we can solve it.

Together with the rest of the committee I've worked to make the running of the organisation more transparent and accessible to ordinary members. This means that minutes are made available on the accu-members mailing list and that attendance at committee meetings for non committee members has been made easier.

Maybe you have some ideas of things the ACCU should be doing? If so stand up and try to make them happen by putting your name forward for committee.

One committee post that will need a volunteer for the coming year is that of Membership Secretary as Mick Books intends to stand down. Mick has been doing an excellent job – but feels it is time to pass on the job. He is willing to remain on the committee to assist his replacement over the coming year and has provided the following responsibilities of the membership secretary:

- Consult website once per month to estimate number of journals required at next printing, and notify the production editor.
- Consult website once per month to retrieve mailing info for the journal, and forward to the distributor.
- Receive excess journals each month, and store up to a year's worth.
- Consult bank statement once per month and enter standing order payments into the website admin interface.
- Chase any standing order underpayments.
- Send journals out to members who have missed an issue.
- Send journals out to prospective members, groups or conferences that request them.
- Keep an eye on the stream of automated renewal and joining payments.
- Answer enquiries to accumembership@accu.org, typically:
  - prepare invoices and receipts
  - resolve payment troubles
  - handle address changes
  - advise on the benefits of membership
- Keep some stats on the size and makeup of the membership, reporting back to the association.
- Prepare reports for the treasurer on the donations that members make when they join or renew.

The AGM and the Proposed New Constitution

Giovanni Asproni
secretary@accu.org

As many of you already know, the time for the next AGM is approaching fast. It will be held on the 13th of April 2013 at the Marriott Hotel City Centre in Bristol, UK. The main deadlines are the following:

- Notice of the Annual General Meeting shall be communicated to the Membership at least 42 days before the Meeting
- Notices of Motion, duly proposed and seconded, must be lodged with the Secretary at least 14 days prior to the General Meeting (however, attendees can propose motions also during the meeting)
- Nominations for Officers and Committee members, duly proposed, seconded and accepted, shall be lodged with the Secretary at least 14 days prior to the General Meeting (however, nominations can also be made during the AGM)
- Amendments and additions to the ACCU constitution will be notified to the secretary no later than 42 days before the General Meeting for which they are proposed
- Amendments and additions will be notified to members by the secretary no later than 28 days before the General Meeting for which they are proposed
- No changes to constitutional amendments and additions will be allowed after the notice to members has been issued
- The 42 days deadline is the 2nd of March and the 28 days deadline is the 16th of March.

The members will be sent the notification for the general meeting by email using the information in the members database, so please make sure your contact details are up to date. We will send it also to the accu-members mailing list.

That said, the most important motion will be the one about the constitutional changes to allow members that cannot attend the AGM (or another general meeting) in person to express their vote for the officers elections and for the motions that have been proposed.

That goal is quite simple, but it requires many deep changes to the current constitution, and also to the way the elections and AGM are run. At the time of writing, there is a conversation on accu-members on how to make the proposed draft better, and I encourage every member to take part to the discussion.

There are also some other constitutional motions for different purposes. You can find them all in this page http://accu.org/index.php/members/constitutional_motions in the members section area of the ACCU site (you need to be logged in to see it).

Please remember that for the constitutional motions, if you cannot attend the AGM, you can also vote by proxy, either by nominating another member to cast a vote in your behalf–you will need to send me an email at secretary@accu.org with the name and membership number of your representative–or by emailing me your vote at secretary@accu.org. Either way, you need to notify me before the AGM starts.

Finally, I encourage everybody to think about the committee and the work done by its members. This is your chance to decide to nominate again the ones you think did a good job, or to nominate new ones if you like. You can do that either during the AGM, or by sending me your nominations at secretary@accu.org.

I hope to see you in Bristol.