the magazine of the accu

www.accu.org

{cvu}

Volume 30 • Issue 2 • May 2018 • £4.50

Features

The New C++ Interview
Pete Goodliffe

Libraries, Console Apps and GUIs
Chris Oldwood

On Quaker's Dozen
A Student

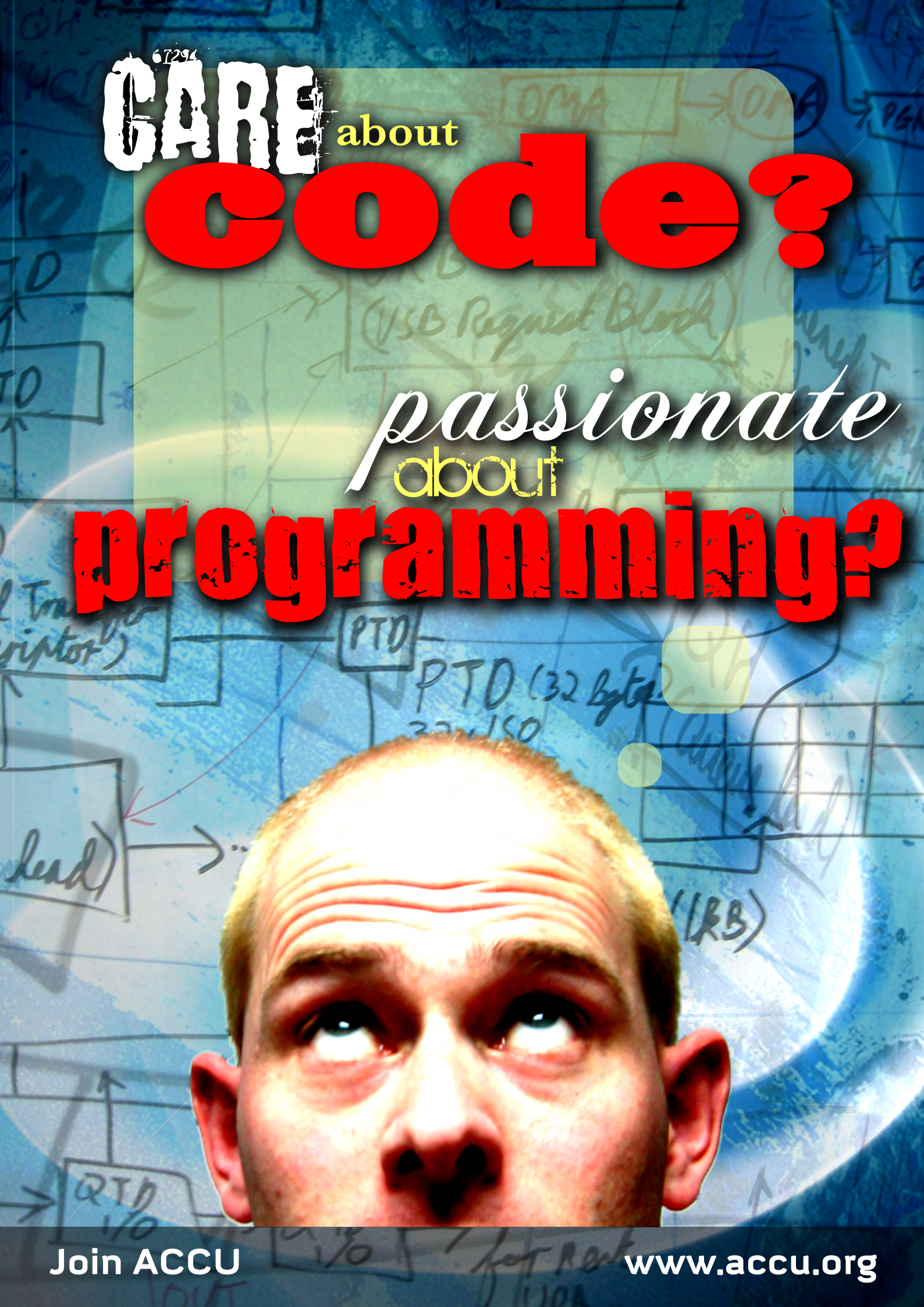
Writing a Wayland Server Using Mir
Alan Griffiths

ACCU: The Early Days
Francis Glassborow

ACCU Conference 2018: Trip Report
Kris van Rens

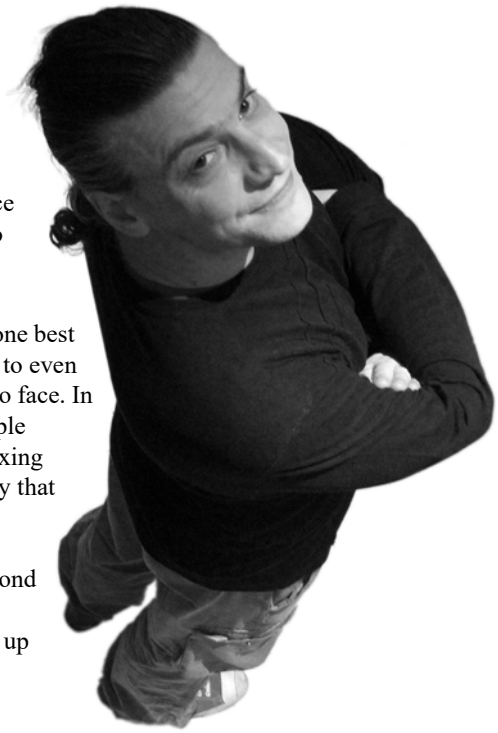
Regulars

ACCU Regional Meetings
Code Critique
Members' Info


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

{cvu}
Volume 30 Issue 2
May 2018
ISSN 1354-3164
www.accu.org

And another thing...



Those of you who attended ACCU's 2018 conference in Bristol may have witnessed me giving a lightning talk about interview techniques. A 5 minute rant is obviously not sufficient to cover all the misgivings I have about how tech interviews are commonly conducted (and I've used this space previously, too!). One aspect I didn't go into was the practice of having several rounds of interview in order to finally reduce the list of candidates down to the 'best of the best of the best'.

Basic statistics suggests that, assuming there is one best of the best, as a hiring company you're unlikely to even see their CV, never mind get to meet them face to face. In addition, expecting candidates to undergo multiple rounds of lengthy interviews, tournament (or boxing match) style runs two major, related risks. Firstly that your actual best candidate decides they aren't prepared to keep returning, or they get another position with someone else. This leads to the second risk: that you end up filtering not for technical proficiency or team best-fit, but instead just end up with the people with the highest tolerance for interviews.

The interview contest itself seems also to have become a test of arcane knowledge often unrelated to the job being hired for. Low-level algorithm design can be fun, but very few working programmers use that knowledge routinely. I know a few that do, but not for their day job – they do it for no pay, in their spare time. In an analogy with the actual practice of programming, high-level languages and rich libraries allow us to *think* at a higher level of abstraction. I would like to see interviews that filter for people at a higher level too. In practical terms, I would much rather be working with people who understand and care about automated testing, continuous integration and delivery of robust, reliable software, than with people who know nothing of these things but have a deep understanding of red-black trees, and are able to implement them with pen and paper.

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

13 Standards Report
Emyr Williams reports on news from the world of Standards.

15 Code Critique Competition 111
The results from the last competition and details of the latest.

22 Local ACCU Meetups
Frances Buontempo reports from London and Bristol.

23 ACCU Local Group
Jason Spencer reports on a recent visit to ACCU Oxford.

REGULARS

24 Members
Information from the Chair on ACCU's activities.

FEATURES

3 Libraries, Console Apps and GUIs
Chris Oldwood compares the use of different application interfaces.

6 ACCU: The Early Years (Part 2)
Francis Glassborow continues his look at the history of ACCU.

7 The New C++ Interview
Pete Goodliffe introduces a new way to test programming skills.

8 ACCU Conference 2018: Trip Report
Kris van Rens shares his experiences from ACCU 2018.

9 On Quaker's Dozen
A student examines the Quaker's Dozen wager.

10 Writing a Wayland Server Using Mir
Alan Griffiths explains the basics of a new X11 replacement API.

SUBMISSION DATES

C Vu 30.3: 1st June 2018
C Vu 30.4: 1st August 2018

Overload 145: 1st July 2018
Overload 146: 1st September 2018

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

Libraries, Console Apps and GUIs

Chris Oldwood compares the use of different application interfaces.

Generally speaking, software tools come in all shapes and sizes from one-line shell scripts [1] through to office 'productivity' suites and beyond. If you're someone who uses a computer to do a non-IT related job, you're more likely to sit at the far end of the spectrum plugging away on a GUI-based line-of-business (LOB) application interspersed with copious context switches to and from your email client, word processor and spreadsheets.

On the other hand if your primary focus is within IT itself, perhaps as a system administrator, tester or programmer, you're more likely to use a selection of tools that runs the full gamut. As I've illustrated in a past episode of this column about the 'simple' task of finding some text [2], the nature of the tool you use also takes on a different form depending on the exact nature of the problem. Not only does the choice of tool depend on what outcome you're looking for but it also may be driving how much you know about the problem domain itself. For example, a GUI based XML document viewer might allow for more freedom of navigation whilst also hiding superfluous details so that you get to see the overall document shape, whereas a simple flat text editor or command window often gives you all the detail until you figure out how to reduce it.

Jeffrey Snover [3], one of the principle architects of PowerShell, once said the difference between GUIs and command line tools is about learning versus automation. For a non-programming oriented audience, I've found that to be a very useful distinction; however, for those people like myself who have a penchant for programming and home-grown tools, I'd like to extend his definition slightly further:

- Graphical applications allow exploration
- Command line tools support automation
- Libraries enable customisation

As we shall see later, the addition of libraries into the mix has a knock-on effect on how we think about designing systems from the perspective of supporting them later.

Graphical applications

Like many programmers, I found the graphics capabilities of computers intriguing and it inspired me to learn more about them. The ease with which you could draw a graph on an 8-bit home computer back in the 80s rather than stare at a table of numbers was one of the reasons I got into this game. As that moved on to animating sprites and eventually 3D models it was no surprise that my first professional programming gig involved working on graphics applications – a drawing package, desktop publisher and associated graphics tools.

Although the dominant force for graphical applications, at least in the early days, was probably down to the promise of WYSIWYG for media creation, it also provides a way to represent more abstract concepts in a visually pleasing way that affords the user a fresh perspective on the task at hand. For example, although folding text editors and syntax highlighting might add some extra pizazz to a mundane XML text document and allow the basic structure to shine through a little, a more dedicated tool that uses icons for the tree structure and separates out the element's attributes might make comprehension easier. Throw in a find first / next feature that supports XPath expressions and you can explore the document in a different way that may gel where a manual navigation approach has meant you can't see the wood for the trees.

Graphical diagnostic tools can also be less intimidating to a more casual user and may even stand-in where the primary product fails. I've had a GUI based integration test harness get shipped to beta testers because it provided a means of working around temporary failings in the core product. It's also not the only time I've seen test harnesses take on a life outside the development team [4]. Any tool which is written to support development is almost certainly going to be useful to support the production system too, although you way want to add some extra features to try and minimise embarrassing mistakes [5].

Naturally we're not just limited to classic thick client applications here either. Services can expose an endpoint for control and monitoring, perhaps through an HTTP endpoint on the side, and the meteoric rise of the browser means that client-side HTML and JavaScript has its uses. If the team has a particular preference for one brand of browser then plug-ins are another way of leveraging someone else's product to put a face on your custom functionality.

Command line tools

GUIs may be great for interactive use but are generally a nightmare to automate. When you're trying to automate tasks, you really need tools specifically designed for automation. Even though a UI like TortoiseGit/SVN provides access to its feature via a command line process called TortoiseProc, you just know that a message box popping up on an unattended server is going to ruin your day. Then you're into the arms race of trying to download and run a tool which watches for (and automatically presses the 'OK' button on) errant message boxes when they appear.

Even OLE Automation which, as its name suggested, promised to allow GUI based office tools like Word and Excel to be programmatically driven didn't quite work as reliably unattended as was perhaps hoped for. In the finance world where Excel is the ad hoc pricing tool of choice, many man-years of development time has been wasted trying to automate spreadsheet calculations (server-side) on the expectation that that task was easier (or cheaper) than getting a developer to code up the same logic up in a 'real' programming language. These days the Excel 'calculation engine' is more accessible but one wonders what the total cost of ownership has been for some of those earlier Heath Robinson approaches and whether it really paid off in the end.

The UNIX philosophy of small, simple command line tools, composed by passing textual output from one to another, is still very much alive and well today. Driving these tools effectively is often for the more seasoned user as finding the right incantation of switches and syntax might only be obtainable with a solid read of the manual (or Stack Overflow) and plenty of examples. Although the industry is slowly converging on common patterns of switch indicators and naming conventions, there are still far too many cases where case-sensitivity and the ability (or not) to merge multiple single character switches will trip you up. Go and PowerShell, with their single dash and long name convention, are notable exceptions

CHRIS OLDWOOD

Chris is a freelance programmer who started out as a bedroom coder in the 80's writing assembler on 8-bit micros. These days it's enterprise-grade technology in plush corporate offices. He also commentates on the Godmanchester duck race and can be easily distracted via gort@cix.co.uk or @chrisoldwood

to the typical dual switch approach (i.e. short and long form). Of course on Windows you've always had to play the game of 'slash or dash' as you're never quite sure if you're dealing with a native Windows tool or a UNIX port. The rise of PowerShell and the new Windows Subsystem for Linux (WSL) will hopefully continue to educate Windows programmers on the correct choice of command line switch character.

The new-fangled Continuous Integration / Delivery / Deployment movement has definitely helped to promote a more automated software development environment and therefore stitching together sequences of tools either with pipelines or through scripting is a more-essentialskill than ever. Even though there are a number of products out there that aim to give you a drag'n'drop approach to creating and managing your build pipeline, I still find a single build script that can run on both a developer's workstation and CI server side to be preferable to two different build systems. In my experience, the developer one is only a pale imitation of the real one and therefore awkward to reproduce and debug, and leads to 'poisoning' of the build machine as more and more junk gets installed.

Despite the plethora of desktop products which provide you with a GUI for automating all manner of tools – GUI or command line based – which are no doubt a boon for the less tech savvy, a shell still seems to be the most popular choice for sequencing operations by the IT crowd. We shouldn't be surprised as scripting is programming after all, just with much 'heavier' functions. Ultimately, unless something is only going to be run on the machine you developed it on, you need to consider what capabilities the target host has and that will likely be the de facto shells. Factor in trying to get the licensing sorted for any 3rd-party product and the path of least resistance wins almost every time.

Libraries

In the traditional taxonomy of software tools, which is what Jeffrey Snover was no doubt aiming at, libraries do not feature as they were never really usable out-of-the-box. Native code libraries packaged in their static library form are intended to be consumed at build time, which means someone needs to write a host application for them. On the other hand, dynamically linked libraries could be consumed directly if you knew what entry point to poke, what arguments to pass and how. On Windows, this is possible to a degree with the RunDll32 utility but it's rarely used in practice as little things like calling conventions make this a dangerous game to play. You could invoke a `void` function taking no arguments fairly safely but anything else was tantamount to running around with scissors.

The Component Object Model (COM), along with a few other technologies like Visual Basic and the Active Scripting Engine, allowed libraries to become scriptable 'components' by standardising the calling convention and adding some metadata to make them a little more discoverable; not quite `--help` territory but a little better. The introduction of .Net and PowerShell took the idea a step further, although one shouldn't discount the various other Windows offerings such as IronPython and F# as suitable scripting hosts; it's just that they aren't installed by default.

It might seem as though the humble static library has been short-changed. The problem is that to consume these you need to have a compiler and native programming skills. But that is exactly what customisation is all about – taking the core functionality for a tool, like Git, and sticking your own console application or UI on top of it. Or, as we've already mentioned above with the Tortoise family, you might do both. Git is a prime example because its complexity makes it hard to grasp but the success of the TortoiseGit tool is almost certainly down to bringing familiarity to those used to the simpler concepts of TortoiseSvn.

For languages that provide interop with C, and let's face it the most common ones do, you have the option to provide plug-in shims that allow the native functionality to be exposed through an adapter and consumed by the plethora of modern, non-native (e.g. JVM) based development tools like IDEs and CI services. Pure versions often arrive later but in cases like the Git TFS bridge where the back-end libraries are proprietary, it may be the only choice. It must surely be a testament to the Git client architecture that adapters for so many different VCS products have been produced that give a reasonably seamless experience.

Passing credentials into Git is an excellent example of where the strength of a good library comes in. When dealing with private repositories, the ability to securely drive Git in your build pipeline without exposing your credentials in log files and environment variables was originally quite a challenge as many of the supposedly mature, off-the-shelf choices struggled with this requirement. Throw in the new-fangled 'submodules' feature and it started all over again as we needed to pass different credentials along the submodule pipeline. For a while I seem to remember switching between various Git plugins and command line tools in search for the right combination that worked with our team's 'Enterprise' set-up, which threw in an NTLM-based proxy for extra friction.

Oranges are not the only fruit and these libraries (static and dynamic) aren't the only two kinds either – anyone working in the Perl, Ruby, Python, Node.js, etc. space will quickly point out that they have had a solution for consuming libraries easily for years, although they tend to talk about 'packages' rather than libraries. As we know [6], these can be fragile ecosystems at times if not treated with respect due to the ever growing volume of dependencies that get pulled in as we struggle to avoid reinventing the wheel.

For development purposes, where I might need a solution to a short lived problem, the reward can far outweigh the risks, but having been on the roller coaster journey of breakages and fixes in areas where you would prefer to have more stable products, there is much solace to be gained from simply copying a single, small binary around. In a production support scenario, I'm even less enamoured by the idea of downloading half the Internet just to run a tool. In a cloud hosted environment where you typically restrict both inbound and outbound network access, this would be a non-starter.

Going back to the UNIX philosophy, one wonders if the notion of 'small' is only considered by some to apply to the amount of functionality it provides, not the size of the ecosystem required to create, distribute and execute it.

Building systems as toolkits

One of the oldest concepts we should be cognisant of when building software systems is that of modular design. Breaking a system down into small building blocks that separate out concerns to try and manage complexity is an on-going battle which programmers face every day. The introduction of lower-levels of testing, e.g. at unit and component level, have definitely had an effect in this area, but another driver you might want to consider for your design is the need for custom tooling.

Systems with plenty of moving parts – databases, queues, services, etc. – have many points of failure, or alternatively, points of diagnostic access. Although it may be built entirely on open protocols and products, that doesn't mean the lowest common denominator tools are the best tools for poking around, e.g. CURL for REST APIs. We build our system out of these building blocks because we have some value to add to them, hence our tooling should add value too. For example an HTTP based API that also needs authentication for its requests soon becomes tiresome as every invocation will require at least two requests – one for the token and one for the actual operation. Wrapping this up in a script is an easy win but if you've built a richer proxy for use elsewhere in your system then stick a simple CLI on top and you have the makings of a deployment smoke test and support tool.

What makes this type of tooling better is that you're leveraging the same well-tested production code in your support toolkit as what you use in the actual system. This really starts to pay off when you need to perform operations that would be considerably more dangerous if you were to hack away at the underlying 3rd party products using their more generalised tools. For example, a database is just an implementation detail of a system and whilst you can poke around with a SQL tool you run the risk of violating invariants of the system that would be enforced if you went

through your system's code. An RDBMS can provide a certain degree of protection from 'tinkering' if you use its referential integrity features to the fullest, whereas document databases and file-systems expect the application to do most of the heavy lifting to ensure the data remains semantically valid.

To give a more concrete example, imagine having a service where you handle versioning of persisted data by upgrading it at read time. Over time, you end up with objects of different versions and therefore lots of code and tests to handle the various incarnations. It can get trickier to make changes as you start to consider all the variants, plus looking directly in the data store during support gets harder as you have to factor in all possible versions into your queries. If you could upgrade all the objects to a new baseline version you could then remove a lot of complexity, i.e. dead code and tests. The production code already exists to do the job; you just need to be able to access it. Although you could add a new private service endpoint you should be able to just host the underlying logic in a simple process container (e.g. scripting host) that can slowly walk the object store and call `Load()` followed by `Save()` for the instances of object that need uplifting.

By building the logic into the production code rather than the support tooling you naturally reduce the risk of a tool getting out of step with the real code and then corrupting something accidentally later. (How many people have tests for their tools?)

Learning a codebase

Aside from their primary purpose – to accomplish certain tasks more easily – custom tools also provide a useful auxiliary function which is that their lower-level focus can help with learning a new codebase. Learning a complex system comprising of many moving parts can be a daunting experience as you can struggle to find a place to start looking. Whilst the natural place might be the UI as the user is ultimately the primary point of focus, with a reporting engine, working backwards towards the inputs can be hard work. The tools won't really help you get an overview of a system but they can be a good way to help understand how part of a system works in isolation. (Obviously a suite of component level tests would provide an even better source of discovery but rarely do you find many tests in between the unit and end-to-end levels.)

In this scenario, I've found command line tools much better for this purpose than GUI based equivalents. What makes tests and command line tools preferable is that they both generally allow you to take a 'straight line' through a piece of functionality. In essence, they gather the inputs, invoke the behaviour, and do something with the outputs – exactly the same kind of approach that makes reasoning about pure functions easier. In contrast, GUIs are inherently event-based and therefore much of the functionality used to prepare the inputs are scattered about in different event handlers with state built up piecemeal before the final push. If the code is not well factored – and we're talking about a custom tool here so attention to detail is less likely – then it can be harder to separate the GUI glue from the production code. For example you may have to keep one eye on the call stack or look at what namespace bits of code belong to to know if you're inside or outside the production code for the component of interest.

Sticking to what we know

In my early days as a professional programmer, my preference would have been to opt for a GUI based approach, which is probably understandable as what I was working on were GUI-based products. Many of the early 'services' I worked on also had a built-in UI to act as a sort of 'administration console' as it was far easier than building in some form of IPC and separate UI tooling back then. It wasn't until a little while later when I started working on more traditional heavily distributed 'headless' services that I started to truly appreciate the power of well factored libraries, command line tools and automation. However, I still built a couple of UI based test harnesses because manual testing was still the dominate approach.

Despite a move into REST APIs for a period, which is clearly a server-side role, I've still had to keep my eye in on the UI based tooling as showcasing the API features both to the product owner and wider shareholders is much easier with a more user friendly approach [7]. While using classic REST tools and Swagger work up to a point, the moment the 'user' journeys involve any kind of authentication the implementation details start to dominate the proceedings. A 'demo UI' can also act as an exploratory testing tool for the API too so it has value outside of showcasing new features.

What is becoming more apparent to me, however, is that it is all too easy to stick to what you already know and create a tool which serves your immediate needs without considering its audience and whether more than one user interface might be desirable because the same the same tasks straddle the lines of analysis, development, testing, deployment and support. It's highly understandable of course because it's probably just a yak you're shaving and a means to a different end than the feature you're supposed to be delivering to your customer. Don't forget though that non-functional user stories have considerable value too and that maybe it's just a matter of prioritisation.

EXIT_SUCCESS

Gerry Weinberg [8] suggests that you don't really understand a tool until you know how to abuse it at least 3 different ways. I don't know if trying to automate Excel on a server counts as one of your three but it definitely qualifies as abuse. One presumes he was talking about the more functional aspects rather than the friction generated by having the wrong form of interface. Either way, it's important that we pay attention to what toolkit we've built for our system as much as the end product we're using it on. And, if we get the design of the system right, the addition of custom tools shouldn't be much of a burden on delivery as the functionality we need to expose will only be a few lines of code away. Our final consideration is then how we're looking to interact with it and whether automation may be desirable, along with learning, exploration, or all those things, but in different ways. ■

References

[1] 'In The Toolbox – Wrapper Scripts', Chris Oldwood, *C Vu 25-3*, http://www.chrisoldwood.com/articles/in-the-toolbox-wrapper-scripts.html

[2] 'In The Toolbox – Finding Text', Chris Oldwood, *C Vu 27-6*, http://www.chrisoldwood.com/articles/in-the-toolbox-finding-text.html

[3] *Wikipedia*, https://en.wikipedia.org/wiki/Jeffrey_Snover

[4] 'From Test Harness to Support Tool', Chris Oldwood, blog, http://chrisoldwood.blogspot.co.uk/2012/11/from-test-harness-to-support-tool.html

[5] 'Afterwood – Honesty', Chris Oldwood, *Overload 139*, http://www.chrisoldwood.com/articles/afterwood-honesty.html

[6] 'Afterwood – Knocked For Six', Chris Oldwood, *Overload 136*, http://www.chrisoldwood.com/articles/afterwood-honesty.html

[7] 'Showcasing APIs – The Demo Site', Chris Oldwood, blog, http://chrisoldwood.blogspot.co.uk/2016/06/showcasing-apis-demo-site.html

[8] *An Introduction to General Systems Thinking*, Gerald Weinberg, 1975 / 2001, ISBN 0932633498

ACCU: The Early Days (Part 2)

Francis Glassborow continues his look at the history of ACCU.

I left off these reminiscences with me being Membership Secretary and the new formed committee agreeing to publish *C Vu* four times a year. A regular publication schedule is important because it helps get material to publish. Deadlines focus the minds of potential contributors.

We had set the date for an AGM for late spring and the publication date for the next issue of *C Vu* to be just prior to the AGM. When the day arrived, I discovered that the next issue was only in draft and had not been printed. I had no desire to face the membership with such an immediate failure and threw enough of a fit to persuade the Chair (Martin Houston) to take the print-out and get enough copies photo-copied to ensure that all attendees at the meeting had a copy.

I was even more annoyed when the editor of *C Vu* said that the next issue would be late because he was going to be in the US in the weeks prior to the scheduled publication date.

New members were going to be finding someone to ask about why they had not had the scheduled issue. Almost certainly that would be the membership secretary, as they would think something had gone wrong with their membership.

After some thought, I turned up to the next committee meeting with an offer that they would be unable to refuse. I volunteered to produce *C Vu* and guaranteed that there would be 6 issues a year and that each issue would be a minimum of 32 A5 pages even if I had to write them myself.

In the event, the smallest issue I ever produced was 48 pages and as time went by the font size went down so that I could publish everything that was offered. I had a strict policy that if a member could be bothered to write, I would find a way to publish. I think our youngest contributor was about 14 and some of the articles were pretty naïve by modern standards but it got people involved. I seem to remember publishing an issue with 80 pages and the average was around 64.

I only once declined to publish an article. That was from someone with a weird idea of mapping the elements of C onto the human brain. I could not make any sense of it and I doubt that the membership would have been able to either. The author was not pleased and accused me of censorship. I just ignored his rantings and eventually heard no more from him.

As I had retired (through stress related ill-health) from teaching, it did not worry me if I sometimes made a fool of myself (like the editorial where I meant to write 'time in lieu' but actually wrote 'time in loo' – spell checkers have limitations) or exhibited ignorance. However, I did have a couple of regular columnists who needed some protection. The Harpist, now dead as a victim of some Middle Eastern conflict, could not write under his own name because his work in the security services prohibited it. Then there was George Wendle (no, that was not his name, and we chose it after searching the nascent internet to ensure that there was no such person (there still isn't, as far as I can check). In George's case, his employer would have started claiming IP over his writing (yes, some employers can be very unhelpful).

At that same auspicious committee meeting where I became editor of *C Vu*, I discovered that any British citizen could attend the lowest level of BSI Committees as long as the convenor was happy with their attendance. Neil Martin was convenor of the C committee and so I started attending those meetings. Soon after, there was the first London meeting of WG21 (the ISO work group for C++, the newcomer on the block). By this time CUG(UK) was broadening out to include some C++, so I arranged to attend that meeting for a couple of days. It was an interesting experience, watching these experts debate language issues. On the second day, I noted that Bjarne Stroustrup was sitting a couple of rows behind me. I had recently read and reviewed the 2nd edition of *The C++ Programming Language*. I thought it would be nice if we offered him honorary membership of CUG(UK) and so went over to him at one of the breaks and introduced myself and made the offer (as a deputy head had told me many years before 'if you do not ask you do not get, and the worst that can happen is that you get a "no"') He accepted the offer. Somewhere in the brief ensuing conversation, I mentioned that I had just finished reviewing the 2nd edition and had found it much more readable than the first. He advised me that writing was a learning experience. He also pointed out that the structure of *The C++ Programming Language* was based on K&R who, I understand, were just down the corridor at AT&T Research.

Somewhere around this time, Will Watts, who was then editor of .EXE Magazine (the '.' Eventually got dropped from the title), contacted me and invited me to try out as a columnist on C. The idea being that I would write 500 words on C and that a member of the European C++ User Group (more about that another time) would write 500 words on C++. What I had not realised at that time was that 500 words meant 500+/- 3. Any more or less and I would have to edit my column to fit. I can tell you that having to write to such strict word counts is a great learning experience and I owe a great deal to Will for taking on an inexperienced writer and helping me hone my writing skills. I continued that column until .EXE stopped publication. I can remember Will letting me know that my current column (which was 2000 words by then) would be the last and his relief when he realised that it was my hundredth column.

Next time I will write about how CUG(UK) became ACCU and how *Overload* came to be. ■

FRANCIS GLASSBOROW

Since retiring from teaching, Francis has edited C Vu, founded the ACCU conference and represented BSI at the C and C++ ISO committees. He is the author of two books: *You Can Do It!* and *You Can Program in C++*.

The New C++ Interview

Pete Goodliffe introduces a new way to test programming skills.

I've been hiring C++ developers for many years now. There has been much debate about the best (most humane, most fair, most inclusive, most accurate...) way to perform these kinds of technical interview. But I may have the solution.

You want to hire people who…

- know C++
- can solve tricky, interesting problems
- think laterally and logically
- have a sense of humour

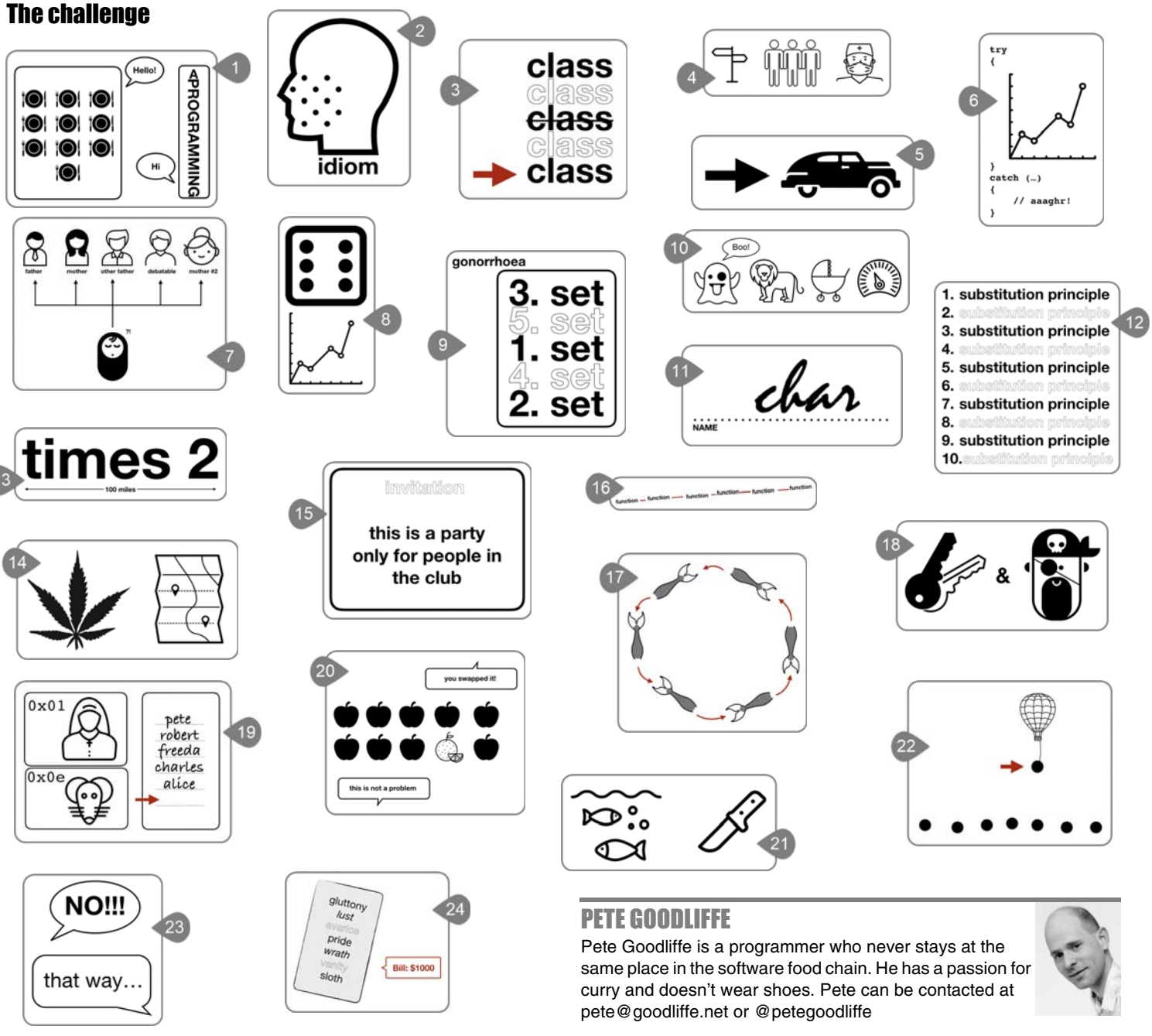
So here, I present (with apologies to your sanity) my latest interview test paradigm. Forget coding tests. Forget inane questions about the number of beagles in Brooklyn. Forget asking for history of your interviewee's programming career.

You can find the perfect C++ coding candidate simply using the magic of **C++ Dingbats**. It works like the old British quiz show 'Catchphrase'. A well-known C++ (or general programming) term is represented pictorially. You need to work out what that term is from the picture. Often you just need to say what you see, or work out the puzzle logically (or, sometimes, cyrptically).

For example, this is a *dangling pointer*. Simple, and obvious, eh? So now you see how it works.

How well can you get on with these interview posers? Are you the perfect C++ interview candidate? Answers on the next page. Don't peek until you've tried to solve the puzzles!

The challenge



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

ACCU Conference 2018: Trip Report

Kris van Rens shares his experiences from ACCU 2018.

Splendid! I'm going to the ACCU2018 conference in Bristol!

Ever since becoming a member of the ACCU in 2016, I have wanted to go. Of course, the ACCU journals are great, but actually meeting the ACCU team, regular writers, as well as hundreds of other like-minded people, is just priceless. I was pleasantly surprised by the family-like open atmosphere, and at the same time by the quality and technical depth of the talks. Sure, I had been to other conferences before, but I've never encountered something quite like this.

The general aim for my visit of course was professional development (or, that's what I promised my employer…). Sharpening my C++ programming skills and understanding. But the conference offered so much more. Aside from all the great in-depth talks on C++, there were loads of talks about other programming languages and programming in general. But then still, there was a plethora of other topics covered. Just to name a few: inclusivity/diversity in the technical field (see [1]), organisational/technical team leadership (or not), human reasoning processes and even a pub quiz. And then besides all the invaluable content of the talks, the proficiency of some of the speakers was just mind-blowing. Very inspirational.

Obviously the keynote talks were single-track talks, but being able to only be present in one place at a time meant missing out on at least four other interesting talks. I'm very thankful to the conference committee for arranging the video recording of (many of) the talks. It will take me quite some time still to soak up all of the content I marked as interesting – which is a lot.

One of the highlights for me was getting a closer look into the kitchen of C++ standardization. A number of committee members were around to give talks, and were grilled on Friday (well, it wasn't that bad). I was pleasantly surprised at how many of the conference attendees had standards proposals in progress or even accepted.

Another interesting (and recurring) topic was our failing education system when it comes to the educating of good C++ programmers. There is education in the general topic of computer science, but that's not quite the same. The result is an ecosystem in which many programmers have become 'dark matter programmers' – generally incompetent and oblivious to professional improvement. In my opinion, it is partly our jobs as improvement-aware professionals to take care of this. So, whenever you've visited a conference/read a book/learned something/etc., tell your colleagues/friends about it! Share with them the links of videos/blog posts/*Overload* articles, cook up a mini-course, whatever suits you, it will make a difference.

The underlying issue partly seems to have its roots in kids of a young age for which insufficient programming teachers are available. And that's exactly where initiatives like Code Club come in. Code Club [2] is the charity organization that organises programming workshops for kids. I joined in on the workshop on Thursday and it was great fun! I'm definitely going to give this a follow-up, if only it were on my own kids (evil grin).

Hand-in-hand with the serious business going on at the conference were the end-of-the-day lightning talks. Ten or so five-minute talks about literally anything, interlaced with light-speed famous scientist quizzes led by Pete Goodliffe. There were serious topics in there too, but most of them were a good laugh.

Then there was the conference dinner, the theme of which was 'magical'. Great food, great atmosphere, great people, Russell 'the grey sorcerer' Winder, and a magician making a fool of us (and himself); what more do you need?

All-in-all, I'd say the experience for me was life-changing. With some of the misery going on in the world, it's sometimes good to realise there are so many people that actually can be utterly lovely to each other ;-)

I surely hope to be there again next year! ■

References

[1] https://github.com/include-cpp/include

[2] https://www.codeclub.org.uk

KRIS VAN RENS

Since meeting his dad's 1983 ZX Spectrum, Kris has been captivated by the wonderful world of programming. In 1995 he learned to program 'Pacman' in x86 assembly, followed by learning C and then C++. He is serious about code quality and works in the Netherlands. Kris can be contacted at krisvanrens@gmail.com

The New C++ Interview (Answers)

1. Template metaprogramming
2. Pimpl Idiom
3. Base class
4. Assignment operator
5. `auto_ptr`
6. Trigraph
7. Multiple inheritance
8. Digraph
9. `std::unordered_set`
10. Boolean parameter
11. `signed char`
12. Liskov Substition Principle
13. `long double`
14. Hash map
15. Private member function
16. Inline function
17. Tail recursion
18. K&R
19. Anonymous namespace
20. SFINAE
21. C#
22. Floating point
23. Object oriented
24. Syntax

Image credits

Maxim Kulikov, Aleksandr Vector, lastspark, DTDesign // Linseed Studio // Creative Stall, Garrett Knoll, Anchor Design // Garrett Knoll, Deemak Daksina S // Kido Chang, Simon Child, WARPAINT Media Inc. // Iconic // Made by Made // Atif Arshad, G // Lil Squid, zidney // Oksana Latysheva, priyanka // iconsmind.com, Fahmihorizon, Andrea Novoa, See Link, Llisole // Cezary Lopacinski

On Quaker's Dozen

A student examines the Quaker's Dozen wager.

The Baron's latest wager set Sir R----- the task of rolling a higher score with two dice than the Baron should with one twelve sided die, giving him a prize of the difference between them should he have done so. Sir R-----'s first roll of the dice would cost him two coins and twelve cents and he could elect to roll them again as many times as he desired for a further cost of one coin and twelve cents each time, after which the Baron would roll his.

The simplest way to reckon the fairness of this wager is to re-frame its terms; to wit, that Sir R----- should pay the Baron one coin to play and thereafter one coin and twelve cents for each roll of his dice, including the first. The consequence of this is that before each roll of the dice Sir R----– could have expected to receive the same bounty, provided that he wrote off any losses that he had made beforehand.

Put in these terms it is self evident that Sir R----- should set himself a constant goal at which to stick, for if it were to maximise his expected prize after the first roll, then it should also do so after every subsequent roll. I explained as much to the Baron, but I suspect that I did not have his undivided attention at the time.

Now, if we label Sir R-----'s roll with x_r and the Baron's with x_b then we can therefore formulate his expected winnings as

$$\begin{aligned} w_k &= \Pr(x_r \geq k) \times \mathrm{E}\left[\max(x_r - x_b, 0) \mid x_r \geq k\right] \\ &\quad + \Pr(x_r < k) \times w_k - 1\tfrac{12}{100} \\ w_k \times (1 - \Pr(x_r < k)) &= \Pr(x_r \geq k) \times \mathrm{E}\left[\max(x_r - x_b, 0) \mid x_r \geq k\right] - \tfrac{112}{100} \\ w_k \times \Pr(x_r \geq k) &= \Pr(x_r \geq k) \times \mathrm{E}\left[\max(x_r - x_b, 0) \mid x_r \geq k\right] - \tfrac{28}{25} \\ w_k &= \mathrm{E}\left[\max(x_r - x_b, 0) \mid x_r \geq k\right] - \frac{28}{25 \times \Pr(x_r \geq k)} \end{aligned}$$

where E[X|C] is the expected value of X given that the condition C holds true.

The probabilities that Sir R-----'s score were equal to any particular value of k are easily figured to be

k	2	3	4	5	6	7	8	9	10	11	12
$\Pr(x_r = k)$	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$	$\frac{1}{36}$

from which we can trivially deduce that the probabilities that it were greater than or equal to any given value of k are

k	2	3	4	5	6	7	8	9	10	11	12
$\Pr(x_r \geq k)$	1	$\frac{35}{36}$	$\frac{33}{36}$	$\frac{30}{36}$	$\frac{26}{36}$	$\frac{21}{36}$	$\frac{15}{36}$	$\frac{10}{36}$	$\frac{6}{36}$	$\frac{3}{36}$	$\frac{1}{36}$

Once Sir R----- had chosen to stick with a roll of x_r, he should have expected a prize of

$$\mathrm{E}\left[\max(x_r - x_b, 0)\right] = \sum_{x=1}^{x_r - 1} \tfrac{1}{12}(x_r - x)$$

where Σ is the summation sign, which we can reorganise into

$$\begin{aligned} \mathrm{E}\left[\max(x_r - x_b, 0)\right] &= \tfrac{1}{12} \sum_{x=1}^{x_r - 1} x_r - \tfrac{1}{12} \sum_{x=1}^{x_r - 1} x \\ &= \tfrac{1}{12} \times x_r (x_r - 1) - \tfrac{1}{12} \times \tfrac{1}{2} x_r (x_r - 1) \\ &= \tfrac{1}{24} x_r (x_r - 1) \end{aligned}$$

Acknowledgement

Courtesy of www.thusspakeak.com

and so, by the very definition of conditional expectations, we have

$$\begin{aligned} \mathrm{E}\left[\max(x_r - x_b, 0) \mid x_r \geq k\right] &= \frac{\sum_{x=k}^{12} \mathrm{E}\left[\max(x - x_b, 0)\right] \times \Pr(x_r = x)}{\sum_{x=k}^{12} \Pr(x_r = x)} \\ &= \frac{\sum_{x=k}^{12} x \times (x-1) \times \Pr(x_r = x)}{24 \times \Pr(x_r \geq k)} \end{aligned}$$

We can make light work of figuring these expectations by noting that

$$\begin{aligned} \sum_{x=k}^{12} x \times (x-1) \times \Pr(x_r = x) &= k \times (k-1) \times \Pr(x_r = k) \\ &\quad + \sum_{x=k+1}^{12} x \times (x-1) \times \Pr(x_r = x) \end{aligned}$$

and proceeding backwards from twelve to two

k	$\sum_{x=k}^{12} x \times (x-1) \times \Pr(x_r = x)$	$\mathrm{E}[\max(x_r - x_b, 0) \mid x_r \geq k]$
12	$12 \times 11 \times \frac{1}{36} = \frac{132}{36}$	$\frac{\frac{132}{36}}{24 \times \frac{1}{36}} = \frac{132}{24}$
11	$11 \times 10 \times \frac{2}{36} + \frac{132}{36} = \frac{352}{36}$	$\frac{\frac{352}{36}}{24 \times \frac{3}{36}} = \frac{352}{72}$
10	$10 \times 9 \times \frac{3}{36} + \frac{352}{36} = \frac{622}{36}$	$\frac{\frac{622}{36}}{24 \times \frac{6}{36}} = \frac{622}{144}$
9	$9 \times 8 \times \frac{4}{36} + \frac{622}{36} = \frac{910}{36}$	$\frac{\frac{910}{36}}{24 \times \frac{10}{36}} = \frac{910}{240}$
8	$8 \times 7 \times \frac{5}{36} + \frac{910}{36} = \frac{1,190}{36}$	$\frac{\frac{1,190}{36}}{24 \times \frac{15}{36}} = \frac{1,190}{360}$
7	$7 \times 6 \times \frac{6}{36} + \frac{1,190}{36} = \frac{1,442}{36}$	$\frac{\frac{1,442}{36}}{24 \times \frac{21}{36}} = \frac{1,442}{504}$
6	$6 \times 5 \times \frac{5}{36} + \frac{1,442}{36} = \frac{1,592}{36}$	$\frac{\frac{1,592}{36}}{24 \times \frac{26}{36}} = \frac{1,592}{624}$
5	$5 \times 4 \times \frac{4}{36} + \frac{1,592}{36} = \frac{1,672}{36}$	$\frac{\frac{1,672}{36}}{24 \times \frac{30}{36}} = \frac{1,672}{720}$
4	$4 \times 3 \times \frac{3}{36} + \frac{1,672}{36} = \frac{1,708}{36}$	$\frac{\frac{1,708}{36}}{24 \times \frac{33}{36}} = \frac{1,708}{792}$
3	$3 \times 2 \times \frac{2}{36} + \frac{1,708}{36} = \frac{1,720}{36}$	$\frac{\frac{1,720}{36}}{24 \times \frac{35}{36}} = \frac{1,720}{840}$
2	$2 \times 1 \times \frac{1}{36} + \frac{1,720}{36} = \frac{1,722}{36}$	$\frac{\frac{1,722}{36}}{24 \times \frac{36}{36}} = \frac{1,722}{864}$

Sir R-----'s expected winnings were therefore as shown in Figure 1, overleaf, and I should have consequently advised him to accept the Baron's wager, provided that he stuck with a score of six or greater upon each roll! □

Writing a Wayland Server Using Mir

Alan Griffiths explains the basics of a new X11 replacement API.

We need a new windowing system

The X-Windows system has been, and still is, immensely successful in providing a way to interact with computers. It underlies many desktop environments and graphical user interface toolkits and lets them work together. But it comes from an era when computers were very different from now, and there are real concerns that are hard to meet.

In 1980, computers were big things managed by experts and, maybe, connected to another computer in the same organization. Today a phone is a computer managed by a non-expert, and connected to the Internet. Then, having floating point co-processor was a feature; now, having a graphics co-processor is required.

In 1980, the cost of developing software was such that any benefit to be gained by 'listening in' on what was happening on the same computer was negligible. It wasn't as though there were billions of computers being used for online banking that would willingly install your program.

Adapting X11 to the new requirements of security and graphics performance isn't feasible. The open source world has settled on a replacement: Wayland. Wayland is a set of protocols and extension protocols that allow applications (clients) to talk to compositors (servers) without many of the problems inherent in X11.

Most open source desktop environments have been based on the X.Org server implementation and a range of different window managers and compositors. All this needs adapting to work with Wayland.

For example, the Gnome project has been adapting it's Mutter window manager to support Wayland; KDE has been adapting Kwin; and so on.

I work on another project, Mir, which in addition to being a compositor aims to provide the building blocks of generic window management and independence from the underlying graphics stack.

In this article I'm going to show you how easy it is to write a Wayland server using Mir.

Building the code

The code in this article needs Mir 0.31 or later. This exists in Ubuntu 18.04 and Fedora 28 both unreleased at the time of writing (but due for release before you read this), or from the mir-team/release PPA (See Sidebar).

It is also useful to install the **`weston`** package as the example makes use of **`weston-terminal`** as a Wayland based terminal application and the Qt toolkit's Wayland support : **`qtwayland5`**.

ALAN GRIFFITHS

Alan Griffiths has delivered working software and development processes to a range of organizations, written for a number of magazines, spoken at several conferences, and made many friends. He can be contacted at alan@octopull.co.uk

On Quaker's Dozen (continued)

Figure 1

$$
\begin{array}{lllll}
w_{12} = \frac{132}{24} - \frac{28}{25\times\frac{1}{36}} & = \frac{132\times25-28\times24\times36}{24\times25} & = \frac{3,300-24,192}{600} & = -\frac{20,892}{600} & = -34\frac{492}{600} \\
w_{11} = \frac{352}{72} - \frac{28}{25\times\frac{3}{36}} & = \frac{352\times25\times3-28\times72\times36}{72\times25\times3} & = \frac{26,400-72,576}{5,400} & = -\frac{46,176}{5,400} & = -8\frac{2,976}{5,400} \\
w_{10} = \frac{622}{144} - \frac{28}{25\times\frac{6}{36}} & = \frac{622\times25\times6-28\times144\times36}{144\times25\times6} & = \frac{93,300-145,152}{21,600} & = \frac{51,852}{21,600} & = -2\frac{8,652}{21,600} \\
w_{9} = \frac{910}{240} - \frac{28}{25\times\frac{10}{36}} & = \frac{910\times25\times10-28\times240\times36}{240\times25\times10} & = \frac{227,500-241,920}{60,000} & = -\frac{14,420}{60,000} & \\
w_{8} = \frac{1,190}{360} - \frac{28}{25\times\frac{15}{36}} & = \frac{1,190\times25\times15-28\times360\times36}{360\times25\times15} & = \frac{446,250-362,880}{135,000} & = \frac{83,370}{135,000} & \\
w_{7} = \frac{1,442}{504} - \frac{28}{25\times\frac{21}{36}} & = \frac{1,442\times25\times21-28\times504\times36}{504\times25\times21} & = \frac{757,050-508,052}{264,200} & = \frac{249,018}{264,200} & \\
w_{6} = \frac{1,592}{624} - \frac{28}{25\times\frac{26}{36}} & = \frac{1,592\times25\times26-28\times624\times36}{624\times25\times26} & = \frac{1,034,800-628,992}{405,600} & = \frac{405,808}{405,600} & = 1\frac{208}{405,600} \\
w_{5} = \frac{1,672}{720} - \frac{28}{25\times\frac{30}{36}} & = \frac{1,672\times25\times30-28\times720\times36}{720\times25\times30} & = \frac{1,254,000-725,760}{540,000} & = \frac{528,240}{540,000} & \\
w_{4} = \frac{1,708}{792} - \frac{28}{25\times\frac{33}{36}} & = \frac{1,708\times25\times33-28\times792\times36}{792\times25\times33} & = \frac{1,409,100-798,336}{653,400} & = \frac{610,764}{653,400} & \\
w_{3} = \frac{1,720}{840} - \frac{28}{25\times\frac{35}{36}} & = \frac{1,720\times25\times35-28\times840\times36}{840\times25\times35} & = \frac{1,505,000-846,720}{735,000} & = \frac{658,280}{735,000} & \\
w_{2} = \frac{1,722}{864} - \frac{28}{25\times\frac{36}{36}} & = \frac{1,722\times25\times36-28\times864\times36}{864\times25\times36} & = \frac{1,594,800-870,912}{777,600} & = \frac{678,888}{777,600} &
\end{array}
$$

On Ubuntu use:

```
$ sudo apt install libmiral-dev mir-graphics-
drivers-desktop weston qtwayland5 g++ cmake
```

Or, if using Fedora, use:

```
$ sudo dnf install mir-devel weston qtwayland5
g++ cmake
```

The 'Mir Abstraction Layer' (MirAL) presents the server-side functionality in a way that makes it easy to use MirAL. There was an introduction to MirAL in *C Vu* 28.2.

To illustrate MirAL I'm going to show what is involved in writing a (very simple) window manager. It runs on desktops, tablets and phones and supports keyboard, mouse and touch input. It will support applications using the GTK and Qt toolkits, SDL2 applications and (using Xwayland X11 applications.

The full code for this example is available on github:

```
$ git clone https://github.com/AlanGriffiths/
egmde.git
$ git checkout article-1
```

Naturally, the code is likely to evolve and inspire additional articles, so you will find other branches, but this branch goes with this article. Assuming that you've MirAL installed as described above you can now build egmde as follows:

```
$ mkdir egmde/build
$ cd egmde/build
$ cmake ..
$ make
```

After this you can start a basic egmde based desktop. By default this will use VT4, so first switch to VT4 (Ctrl-Alt-F4) to sign in and switch back again. Then type:

```
$ ./egmde-desktop
```

You should see a blank screen with a weston-terminal session. From this you can run commands and, in particular, start graphical applications. Perhaps qtcreator to examine the code?

There's very little code needed to get this basic shell running:

```
$ wc -l *.h *.cpp *.sh
   88 egwindowmanager.h
   34 egmde.cpp
  420 egwindowmanager.cpp
   47 egmde-desktop.sh
  589 total
```

The example code

A lot of the functionality (default placement of windows, menus etc.) comes with the MirAL library. For this exercise we'll implement one class and write a main function that injects it into MirAL. The main program looks like this:

```
using namespace miral;
int main(int argc, char const* argv[])
{
  MirRunner runner{argc, argv};
  return runner.run_with(
    {
      set_window_management_policy<egmde
      ::WindowManagerPolicy>()
    });
}
```

Most of the logic belonging to egmde is in the `egmde::WindowManagerPolicy` class. This implements a `miral::WindowManagerPolicy` interface that allows the window management to be customized. It looks like Listing 1.

These are the functions that it is necessary to implement for a minimal shell. I won't reproduce them all here, but Listing 2 should give a flavor.

Listing 1

```
class WindowManagerPolicy : public
CanonicalWindowManagerPolicy
{
public:
  using CanonicalWindowManagerPolicy
    ::CanonicalWindowManagerPolicy;
  bool handle_keyboard_event(
    MirKeyboardEvent const* event) override;
  bool handle_pointer_event(
    MirPointerEvent const* event) override;
  bool handle_touch_event(
    MirTouchEvent const* event) override;
  Rectangle confirm_placement_on_display(
    WindowInfo const& window_info,
    MirWindowState new_state,
    Rectangle const& new_placement) override;
  void handle_request_drag_and_drop(
    WindowInfo& window_info) override;
  void handle_request_move(
    WindowInfo& window_info,
    MirInputEvent const* input_event) override;
  void handle_request_resize(
    WindowInfo& window_info,
    MirInputEvent const* input_event,
    MirResizeEdge edge) override;
  private:
  ...
};
```

The way of the MirAL API

The MirAL API is designed so that it is easy to implement and run a Mir server and compose any special features. One aspect of this is the provision of building blocks that can be tailored by the developer and 'hooked up' in the `main()` function by adding them to the `run_with()`

Listing 2

```
bool egmde::WindowManagerPolicy
  ::handle_keyboard_event(
  MirKeyboardEvent const* event)
{
  auto const action =
    mir_keyboard_event_action(event);
  auto const shift_state =
    mir_keyboard_event_modifiers(event) &
    shift_states;
  if (action == mir_keyboard_action_down &&
    shift_state == mir_input_event_modifier_alt)
  {
    switch (mir_keyboard_event_scan_code(event))
    {
      case KEY_F4:
        tools.ask_client_to_close(
          tools.active_window());
        return true;

      case KEY_TAB:
        tools.focus_next_application();
        return true;

      case KEY_GRAVE:
        tools.focus_next_within_application();
        return true;

      default:;
    }
  }
  return false;
}
```

list. One of these is an 'internal client' – a 'client' that runs in the server process but can use the client APIs to, for example, draw a surface.

The next iteration of egmde will add an internal client that paints a configurable wallpaper. I won't show how that is implemented here, but I will show how it is added to `main()` program (see Listing 3).

The `CommandLineOption` utility does a number of things, adds a configuration option to the command line, process the command line, and calls it's first argument. In this case we pass an instance of `egmde::Wallpaper` implements the function call operator to accept the wallpaper colour.

The `StartupInternalClient` utility takes an internal client object, waits for the server to start and then connects the client to the server, notifying the internal client object of both the client-side and server-side connection so that the server 'knows' which client this is.

By supplying customizations to the `run_with()` as a list we make it easy to ensure the server is initialized before they are used and give the user flexibility in setting these objects up. For example, the `wallpaper` instance can be created and used in a a shutdown hook using `add_stop_callback()` before being used in the `run_with()` list.

This is achieved by declaring `run_with()` to take an initializer list:

```
auto run_with(std::initializer_list
<std::function<void(::mir::Server&)>> options)
-> int;
```

Each of the supplied utilities 'knows' how to integrate itself into the system using the `mir::Server`. User code should not need to do this directly, so part of the MirAL 'abstraction' is to keep this as an opaque type.

This approach has been proven effective by use in more advanced servers such as Unity8.

The capabilities

Mir tries to be agnostic about the window management style. There are example shells in the Mir project implementing several styles: 'floating' windows, 'tiling' windows and fullscreen 'kiosk' windows.

Although it was abandoned by Canonical, the Unity8 desktop shell demonstrates the potential of a shell based on Mir.

Mir is also used for Canonical's 'Internet of Things' kiosk and for the Ubuntu Touch phone operating system.

The limitations

Support for Wayland isn't complete in Mir (nor, to varying extents, other compositors) and it isn't complete in the toolkits used to write applications. However, the commitment and momentum is there. It will happen soon.

Listing 3

```
int main(int argc, char const* argv[])
{
  MirRunner runner{argc, argv};
  egmde::Wallpaper wallpaper;
  runner.add_stop_callback([&]
    {wallpaper.stop(); });
  return runner.run_with(
    {
      CommandLineOption{
        std::ref(wallpaper),
        "wallpaper",
        "Colour of wallpaper RGB",
        "0x92006a"},
      StartupInternalClient{"wallpaper",
        std::ref(wallpaper)},
      set_window_management_policy
        <egmde::WindowManagerPolicy>()
    });
}
```

Other ways to get Mir and use for development

On other versions of Ubuntu

If you want to use Mir on earlier supported versions of Ubuntu then it is available from the Mir Release PPA. To add the PPA to your system:

```
$ sudo add-apt-repository ppa:mir-team/release
$ sudo apt update
```

You can then follow the instructions in the main article. One word of caution though: on Ubuntu 16.04 the toolkit support for Wayland is somewhat dated and the experience is not great. On 17.10 things mostly work.

To remove the PPA from your system:

```
$ sudo ppa-purge mir-team/release
```

On other versions of Fedora

Fedora versions 26 and 27 have an earlier version (1.5) of MirAL, that would require a few (simple) changes to the code to get the egmde example code working. More importantly Mir's support for Wayland in this earlier version is lacking. It is probably best to build your own (see next section).

Build your own Mir

There are instuctions for building Mir on the Mir website: https://mir-server.io/doc/getting_involved_in_mir.html

To my knowledge these work on Ubuntu 16.04 or later, Fedora 28 or later and Debian sid. (If you try it on other distros please let me know how it works out.)

Conclusion

If you are interested in experimenting with writing a shell to support Wayland clients then Mir might be an option. ■

References

egmde: https://github.com/AlanGriffiths/egmde/wiki
Mir: https://mir-server.io/
Unity8 demo: https://www.youtube.com/watch?v=LUxVdURZdRk
egmde demo: https://www.youtube.com/watch?v=e4_SEybCB0M
Unity8 use of MirAL API: https://github.com/ubports/qtmir/blob/master/src/platforms/mirserver/qmirserver_p.cpp#L91
Ubuntu Touch: https://ubports.com/

Standards Report

Emyr Williams reports on news from the world of Standards.

The first ISO C++ committee meeting of 2018 was held in Jacksonville, Florida where work continued apace to work on the Technical Specifications, as well as the next version of C++ which would be C++20. Once again making use of the various trip reports, and with my eternal thanks to the people who've written one, I've been able to cobble together what are the main things discussed. I would like to thank Roger Orr for proofing and correcting where necessary.

C++ 17 - Compiler update

As my previous report stated, C++ 17 has now been officially published. However, it's worthy to note that the latest versions of GCC and Clang both have complete support for C++ 17, apart from a few bugs. And Microsoft's C++ compiler MSVC has significant partial support, but they are making good progress towards full support.

C++ 20 Language

Language support for empty objects. (P0840R0)

While we already have an empty base optimization, C++ 20 will have additional features for the creation of empty objects. This is achieved using the **no_unique_address** attribute. This tells the compiler that a unique address in memory isn't required for a non-static data member. This means that we can compose without overhead and without needing to use inheritance, which can be troublesome as it 'leaks out'. It also means that the data member can share its address with another object, as long as it could have a zero size, and they're both distinct types.

```
template <typename Key, typename Value,
  typename Hash, typename Pred,
  typename Allocator>
class hash_map {
  [[no_unique_address]] Hash hasher;
  [[no_unique_address]] Pred pred;
  [[no_unqiue_address]] Allocator alloc;
  Bucket *buckets;
}
```

In the example above, the **hasher**, **pred** and **alloc** objects could have the same address as the **Buckets** object if their respective types are all empty.

Down with typename (P0634R2)

This proposal removes the requirement to use the **typename** keyword as a disambiguator in some contexts. The proposal is that **typename** is made optional in a context that are known to only permit type names.

Allow structured bindings to accessible members. (P0969R0)

Structured Bindings was one of the last-minute additions to C++ 17, and with use users have come across new issues and ideas to make it better. Timur Doumler, who's a member of the BSi panel proposed a paper to enhance structured bindings. The original wording only allowed binding to public members, however Timur's paper proposed to allow binding to any accessible data members. It will be in C++20, however it is treated as a defect for C++17.

Relaxing the range-for loop customization point finding rules. (P0962R0)

This resolves a set of corner cases for the range-based for. At present, you cannot write a range-based for loop over data that comes through an input stream without writing an adaptor class. While this may surprise a few of us, the paper's author was concerned that the committee have painted themselves in to a corner where such support cannot be added. The issue is that **seekdir::end** is found, and this makes it impossible to find **begin()** and **end()** with any degree of certainty.

The paper also provides a code sample that wouldn't work in C++ 17, however it would work in C++ 20.

```
#include <sstream>
#include <iterator>
struct X : std::stringstream
{
  // do some stuff here
};
std::istream_iterator<char> begin(X& x)
{
  return std::istream_iterator<char>(x);
}
std::istream_iterator<char> end(X& x)
{
  return std::istream_iterator<char>();
}
int main()
{
  X x;
  for(auto&& i : x)
  {
    // do stuff here
  }
}
```

Parallelism TS v2 Complete, sent for national body balloting.

The new version of the Parallelism TS was completed, and a draft has been sent for national body balloting, so I expect that will be discussed at the next BSi Panel meeting in London.

Attributes for [[likely]] and [[unlikely]] (P0479R4)

One of the things compilers have been doing for quite some time now, is branch-execution optimisations, but it has tended to be non-portable. One of the new things coming in C++20 is the ability to mark which is the likely branch to be executed, and it will give that information to the optimiser. So for example:

```
if(foo < bar) [[ likely ]] {
  // do something neat here…
} else {
  do_other_thing();
}
```

You can also tell the compiler what is not usually executed, so for example:

```
while( foo > bar ) {
  [[unlikely]] create_widget();
}
```

EMYR WILLIAMS

Emyr Williams is a C++ developer who is on a mission to become a better programmer. His blog can be found at www.becomingbetter.co.uk

This can be used for any scenario where branching occurs, such as a switch statement, a nested if statement, or any of the loops.

There was discussion about the name, since the attribute can also be used for statements executed when rare events occur that needs the faster path. So that's worth bearing in mind.

Coroutines (P0912R0)

The proposal to merge coroutines was delayed, pending papers giving feedback from the TS, especially the one from Google, however this will be visited again in the next meeting in Rapperswil.

New work item proposals created

There was a move to direct the Convener, to request a New Work Item for a Technical Specification on "C++ Extensions for Reflection" and create a working draft with "Static Reflection" (paper p0194r6) as its initial content.

Symmetry for <=> (P0905R0)

The idea behind the paper was to make operator spaceship symmetric, so that when `a <=> b` is well formed, then `b <=> a` should also be well formed and have the complementary semantics. This is helpful for mixed-type comparisons where the operator was only defined in one direction.

Library Evolution Working Group

Much work and effort was made to advance the case for using concepts and modules in the standard library. The Standard Library Concepts paper (p0898r0) was sent to the Library Working Group for a wording review.

The group also worked on a plan to merge the Ranges Technical Specification Version 1 in to C++ 20. So far there hasn't been a consensus on where range-based algorithms should go just yet. There was previous discussion in regard to placing them into a new namespace (std2) however it looks more likely that they will be placed in `std::range`s.

C++ 20 Library

A <version> header (P0754R1)

This would be a new header file with implementation-dependent version information. The information that could be found in this file, may well be different in each implementation and be specifically related to the vendor. This will replace the <ciso646> header file, as it was being hijacked by some vendors as a good place to put vendor-specific defines, which went against the original purpose of the header.

Comparison of Unordered Containers. (P0809R0)

At present, the comparison of unordered containers causes undefined behaviour unless Hash and Pred have exactly the same behaviour for both containers. This means that two different hash functions may lead to containers being different.

Major feature developments timeline

Herb Sutter created a timeline of what's coming when, although he was at pains to point out that this was a plan, and not a promise, so it should be treated as speculative and tentative. It is shown in the table below, which uses the following terms:

- CD = Committee Draft
- IS = International Standard
- TS = Technical Specification

Feature	Status	Depends on	Current target (conservative estimate)	Current target (optimistic estimate)
Concepts	Concepts TS v1 published and merged in to IS		C++20	C++20
Ranges (N4128)	Ranges TS v1 published	Concepts	Core in C++20 and rest in C++23	C++20
Contracts (p0542r3)	Proposal		C++23	C++20
Modules (P0678r0)	Modules TS v1 published		C++23	C++20
Coroutines (N4723)	Coroutines TSv1 published		C++23	C++20
Executors (p0761r2)	Proposal		C++23	C++20
Networking (N4711)	Networking TS v1 published	Executors	C++23	C++20
New future and async	Concurrent TS v1 published	Executors	C++23	C++20
Reflection (p0194r5)	TS Working Paper		TS in C++23 timeframe and IS in C++26	TS in C++20 timeframe and IS in C++23

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?

Code Critique Competition 111

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.

Note: If you would rather not have your critique visible online, please inform me. (Email addresses are not publicly visible.)

Last issue's code

> I've written a simple program to print the ten most common words in a text file supplied as the first argument to the program. I've tried to make it pretty fast by avoiding copying of strings. Please can you review the code for any problems or improvements.

What would you comment on and why?

Listing 1 contains the code. (Note: if you want to try compiling this on a pre-C++17 compiler you can replace `string_view` with `string` and most of the issues with the code remain unchanged.)

Listing 1

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string_view>
#include <unordered_map>
#include <vector>

int main(int argc, char **argv)
{
  std::unordered_map<std::string_view, size_t>
    words;
  std::ifstream ifs{argv[1]};
  std::string ss{
    std::istreambuf_iterator<char>(ifs),
    std::istreambuf_iterator<char>()};
  auto *start = ss.data();
  bool inword{};
  for (auto &ch : ss)
  {
    bool letter = ('a' <= ch && ch <= 'z' ||
      'A' <= ch && ch <= 'Z');
    if (inword != letter)
    {
      if (inword)
      {
        std::string_view word(
          start, &ch - start);
        ++words[word];
      }
      else
      {
        start = &ch;
      }
      inword = !inword;
    }
  }
```

Listing 1 (cont'd)

```
  std::map<size_t, std::string_view> m;
  for (auto &entry : words)
  {
    auto it = m.lower_bound(entry.second);
    if (it != m.begin() || m.empty())
    {
      m.insert(it,
        {entry.second, entry.first});
      if (m.size() > 10)
      {
        m.erase(m.begin());
      }
    }
  }
  for (auto &entry : m)
  {
    std::cout << entry.first << ": "
      << entry.second << '\n';
  }
}
```

Critiques

Paul Floyd <paulf@free.fr>

There's been some discussion on the mailing list about accessing C++17 compilers. This is indeed fairly difficult.

[Ed: I apologise for the trouble I caused by using C++17 features in this critique.]

For the most part I don't use Windows for compiling so I haven't tried any compilers on that platform.

On macOS, the current XCode (9.2, with Apple clang 9.0.0) was able to compile it using -std=c++1z.

On Linux, I build clang and GCC regularly, both the SVN head versions. Early on I gave up trying to build libc++, so I couldn't compile this code with clang, though I admit I gave up quickly. GCC, given enough options, does the trick. I was also able to build the code on Solaris, again with GCC built from source.

For those of us that like to more than just a second opinion, I was also able to build it on FreeBSD. It wouldn't compile with clang 4.0.0 that is packaged on FreeBSD 11.1 but it compiled OK with clang 7.0.0 built from source.

Wrapping up on the language versions, when compiled with `std::string` instead of `std::string_view` I see about a 10% performance degradation. That means `string_view` is quite a nice improvement – the code is essentially the same (no obscure optimisation tricks) for a non-negligible speed gain.

Getting back to the code.

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

clang complains about a lack of parenthesis in the and-or-and expression for ascii letters. GCC complained that `argc` is unused. Nothing too serious.

There is no check that more than one argument has been provided. The dynamic analysis tools that I tried didn't complain, but this should be fixed for production code.

For my testing with a text file, I used a small file containing some random French and then the text of *Tom Sawyer*.

I have a big beef with the following code:

```
bool letter = ('a' <= ch && ch <= 'z' ||
  'A' <= ch && ch <= 'Z');
```

Is this an early word counting Brexit? This treats any non-English ascii letters as separators, so "près" is considered as "pr" and "s"

`std::isalpha` would be better. I did try fiddling with `LC_ALL` and `std::locale`, but didn't manage to get it to work. Unfortunately there is a price to pay for this (and a heavy price if internally all of the strings need to be converted to a unicode representation). When I switched to using `isalpha`, the Valgrind callgrind instruction count went up from about 162 million to 181 million.

There are two possible approaches for splitting the lines into words.

- Look for the letters and assume that everything else is whitespace or punctuation.
- Look for the whitespace and punctuation and whitespace, and assume that everything else is a letter.

I prefer the second approach as there is a lot less punctuation and whitespace than there are letters. It also makes it easier to include things like apostrophes so that "fo'c's'le" gets treated as one word and not 4.

I'm not sure if it is intentional, but the `m` reverse map from word counts to words is a plain map. This means that for any words that have the same word count, only the last one will be recorded.

Moving on to performance. I don't think that it's a good idea to just look at some code and then make changes and hope that it will run faster. What is the performance requirement? You need to know how big the problem is. Is the text going to be 1kbyte, 1Mbyte 1Gbyte or more? And how big is the expected vocabulary in the text? 1000, 100,000, 10,000,000 words? Is the vocabulary fixed in advance? If it is fixed, then I would suggest using a perfect hash and a lookup table. Does the text have a special distribution (like with medical or legal vocabulary), or does it just a 'plain text' distribution?

My expectation is that the bottleneck will be the time taken to read the input file. Callgrind confirms this: 80% if the time is spent in `ostreambuf_iterator::__pad_and_output`. That probably means that any code changes will be 'premature optimisations' and the most fruitful avenue would be to use something like asynchronous I/O. I would expect that the word count map 'words' will have an upper limit in the region of 10 to 20 thousand elements.

I think that it is a waste of time to limit the reverse map of counts to the top 10 words. Even if the input text is large, say 600,000 words (a doorstopper like *War and Peace*), then the largest size of the reverse dictionary will only be around 1100 elements. (The worst case for the 'words' map is that it contains entries with counts of 1, 2, 3, … . The sum of these counts is that for an arithmetic progression, n/2(n+1)). This means that the size of the reverse map is O(Sqrt[Word Count]), and consequently will never be very large.

If performance really is crucial, then it will probably be faster to replace `m` with a table e.g., a `std::vector`. The vector should be resized to 10, and contain the running top 10 words in descending order. For the map, it takes around 3 comparisons to determine whether to insert the word. With a table you just have to compare with the last element. If it is less than or equal, do nothing. If it is more, do a binary search and insert the new word. I expect that insertions will be very rare.

It would be possible to replace the two maps with a single 'bimap'. However, I would advise against doing that if performance is important. It may simplify the code, but my experience is that it doesn't do the performance any favours.

James Holland <James.Holland@babcockinternational.com>

I am afraid that I don't know the finer points of `std::string_view` and so I have replaced them with `std::string`. I hope others will be able to provide help and advice on this topic.

The main problem with the student's code is that should more than one word occur the same number of times within the file, only the last one encountered will be recognised. This is because the map the student is using can only map the number of occurrences to a single word. When inserting a word into the map, any existing word with the same number of occurrences will be overwritten and, therefore, lost. Fortunately, this situation is easily remedied. All that needs to be done is to replace the `std::map` with an `std::multimap`. When this is done, all words will be added to the `std::multimap` and so will be counted. It is interesting to note that, with the amended program, of the words that occur equally in tenth position, it is not possible to predict which will be printed if their number, plus the ones that occur less frequently, exceed 10. This is because the order in which the words are stored within the `std::unordered_map` is not defined. This is the nature of an unordered map! This behaviour may or may not matter for the student's application. A more sophisticated version of the code could display which words occur equally often.

The student's code makes use of quite a complicated expression to determine whether a character is an upper or lower case letter. There is a function available to determine this, namely `isalpha()`. This function returns a non-zero value if the character is an alphabetic character and zero otherwise. This function should be used in preference to the student's code as there is no guarantee that upper or lower case letters are contiguous.

The student has devised a clever way of separating the words within the file. It would be difficult to find a more efficient method, I suggest. The code is quite difficult to understand, however. I wonder if a simpler method exists, perhaps taking advantage of an existing library. From my brief investigations, it would appear that most libraries that extract words (or tokens) from strings are defined in terms of what constitutes the delimiters between words. In contrast, the student's code defines what a word is. This makes selecting a suitable algorithm a little difficult. Regex is very flexible and so is a likely candidate. In this case, the separator is anything that is not a series of alphabetic characters. This can be realised in only four lines of code (as shown in the listing below), compared to the student's 12 (depending on how they are counted). But what about efficiency? My measurements suggest that for my test file the student's code takes about 6 ms to execute whereas the code using regex takes about 30 ms. I did say the student's code would be hard to beat. I suspect regex was designed to crack tougher nuts than this and is not particularly fast for simple cases. Let's now have a look at how the most frequent 10 words are obtained.

Again, the student has developed a bespoke algorithm to extract words from the unordered map and insert them into an `std::map`. This is a clever design that takes a little study to determine how it works. Some experimentation would be required to assess its efficiency and whether it could be improved. I offer a more direct approach that makes use of the standard library. It may not be as fast as the student's code but is simpler to understand, I suggest. The idea is to copy the information in the unordered map to an `std::vector` and then to perform a partial sort of the vector to obtain the 10 most popular words which are then printed on the screen. It is interesting to note that there are no explicit loops or `if` statements in the code I offer.

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <unordered_map>
#include <vector>
#include <regex>
```

```
int main(int argc, char **argv)
{
  std::ifstream ifs(argv[1]);
  std::string
    ss{std::istreambuf_iterator<char>(ifs),
    std::istreambuf_iterator<char>()};
   std::unordered_map<std::string, size_t>
     words;
  std::regex separator("[^[:alpha:]]+");
  std::sregex_token_iterator word(ss.cbegin(),
    ss.cend(), separator, -1);
  std::sregex_token_iterator end;
  std::for_each(word, end,
    [&words](std::string w){++words[w];});
  std::vector<std::pair<std::string, size_t>>
    v(words.begin(), words.end());
  const int top = 10;
  std::partial_sort(v.begin(),
    v.begin() + top, v.end(),
    [](auto a, auto b){
      return a.second > b.second;
  });
  std::for_each(v.begin(), v.begin() + top,
    [](auto p){std::cout << p.second << ": "
      << p.first << '\n';});
}
```

[Ed: this is problematic with fewer than 10 distinct words.]

In the case where efficiency is important, further work could be carried out to find the most efficient method of performing the various functions of the code. Ensuring strings do not reallocate memory while characters are added to them would, I suspect, be something worth investigating. It is important to time the execution of the code as trying to guess its efficiency is unreliable.

Balog Pal <pasa@lib.hu>

This is new, starting from code without an attached "it's broken, does weird things" but instead supposedly working and looking for review. So let's play this as an actual work-place review, not using the compiler, just our eyes and minds.

First I just look from afar: noticing that all we have is **`main`** and all the code is sitting there. I guess that would break most usual style guides and maybe incite shock to some reviewers. Not me. Plenty of the previous CC entries had unnecessary fragmentation, nonsense functions, 1-shot headers, where all the task could be perfectly handled in just **`main`** or 1–2 functions tops. I postpone the verdict for later, but at glance this one is okay to be like that: no repeated stuff that makes extraction mandatory and the sub-parts do not really form abstraction. I mean they do, but not on the level that is needed to improve readability. We have four parts that are trivial to see. And even if they were not, a single line of comment would do a better job than a function, that will add noise and extra need to pass the state. So it is not suggested now. It still can be easily done later if a reason emerges, like someone is willing to write a test case for some part.

The other thing I note at a glance that we have a nice alphabetic list of standard includes but no **`using namespace std`**; or even some **`using`** declarations. In large projects, the directive is frowned upon for good reason. Especially if the project is old and has parts written before the standard. For this one, I don't see a single reason not to have it and get rid of those ugly **`std::`** prefixes.

Still at glance I gather understanding on how the problem is solved and implemented: part 1 reads the file into memory, part 2 isolates the words and builds a map of word/word-count, part 3 selects the top 10 entries with the biggest count into another collection and the final part outputs that collection. This looks like a sensible plan that is fit as a solution, so we can move on to the details. Just have to note a stray line: the first in **`main()`** belongs to part 2 and shall be moved down to next section.

Part 1, where we load the file content into memory. That raises my first concern. The preamble just mentions 'text file'. On a live review, I start asking for the missing info on the intended input, execution environment and consequences of being unable to provide an answer. As the file may be large enough not to fit in the address space, let alone the available memory. And if it fits, but just barely, it may use enough so our further collections will not fit, or lead to poor performance. If we get the answer "yeah, we are on 64-bit platform with 16G memory and the intended files are few megs, 100M tops, if we run out of memory it's operator error", we can continue. Otherwise I'll require a different approach where parts 1 and 2 are put together and words are extracted reading small chunks. Using **`string`** instead of **`string_view`**, but the rest may fly. Review aborted and waiting a second round.

Approving the idea of having the file in memory still leaves us with the lines implementing that. First we just used **`argv[1]`** before checking **`argc`**, having UB if launched without argument. We did not check if the read was successful, that is rude. And we did not deal with the possibly running out of the memory. The latter throws exception that in the current state would cause **`terminate()`**. That might be considered the expected behavior, but in my book playing nice means catching the problem and clearly informing the user. The easiest way is to just surround the code in a **`try`** block and report either just generic failure or add **`what()`** from the exception too. For the **`argc`** and file error cases, we can make an early exit or throw an exception to be reported in the way just described. For the stream we can even turn on **`.exceptions()`** instead of the check, though I would stay with the check.

That leaves just one remaining note, why we use **`std::string`** here. I see no good reason and suggest **`vector<char>`** instead along the guidelines to have that as default. The following code needs no change for that, just the type of **`ss`**.

And one extra question to the client, as I never use streams this way IRL, is why we used **`istreambuf_iterator`** instead of **`istream_iterator`**, which would be intuitive, then make decisions upon the answer.

Finally onward to part 2 (did we really use 400 words on the 2 lines of source in part 1?). My style is to initialize **`bool`** with **`= false`** instead of **`{}`**. While start does not need an initialization value, that holds meaning only when **`inword`** is true. Guess it was motivated by urge to use **`auto`**. That is my favorite keyword in C++, but here **`char *`** is what we want really. **`letter`** should be **`const`**. And I will ask for the excuse for not using **`isalpha()`** from the standard (that I assume unlikely to pass).

The state machine is sound except for the termination. We use address of **`ch`**, so double check that we run the loop with **`&`**. Flipping **`inword`** is up to style, for myself I'd use **`= true`**/**`false`** in the 2 blocks, but that form also passes. At the map insertion I would not use a temporary, as the initialization should fit fine inside the **`[]`**, but if we do, I'd add **`move()`**. Despite it feeling redundant unless we use **`string_view`** – but it may work better with different key type like **`string`**, and cannot hurt.

Thinking of the termination condition, we reveal a serious bug: if the file finishes in a letter, then we drop the last word. Guess the client tested it only with files having **`\n`** or whitespace at the end. To correct we either need to check **`inword`** after the loop and process the last word duplicating the inner code (avoidable by putting it in lambda) – or dodge the problem by adding a non-letter character to **`ss`** before the loop (blindly or with a check).

Before moving on, it is worth pondering why **`unordered_map`** is good for us. And if it is, we still may need to tweak it by setting the bucket number or something. **`string_view`** as the key type is also worth questioning: if the expected files are like English text, it will have many short words. For those, a string with small-string optimization that can cover most cases may work better. The client did mention performance as a goal. It's not hard to make the key type into a **`typedef`** and put the line with the **`map`** insert into a separate function, with variants for **`std::string_view`** and **`std::string`** and possible others, then measure.

Part 3 where we need to fish out the 10 winners. I wonder if there isn't an algorithm for that, need to check **`nth_element`**, **`partial_sort`**... at

cppreference for the former I spot `partial_sort_copy`. What IMHO describes exactly what we're after. And it uses an Input Iterator for source that fits our map. And can take predicate to compare the count. Having that the client code can be written defensively against this, I would not bet on success, as IMO a simple input with a/2 b/1 would output just a, and if we see one cockroach.... it's probably wiser to move to another neighborhood. Especially if we have an offer at hand.

At part 4 I miss a `const` after `auto` (at least consistent with part 3 :), otherwise fine for me if we change `'\n'` to `endl`, at least I'm confident what the latter does without looking up. The original aim did not state the order it wanted the output. For me it would be more intuitive to have it start with the most frequent and go in descending order. It's easy to iterate the map by `rbegin()`. But if we use the algorithm suggested for part 3 the result will naturally look like that and the current form will be good.

I'm also fine with no `return` at end of `main`. Instead on conclusion I must state that a live interview with actual talk with the client and getting answers instead of guessing is way more fun, and less cumbersome. But still an interesting exercise.

Jason Spencer <contact+pih@jasonspencer.org>

This is a nice program in terms of design and container usage. There are just a few things I'd change for correct behaviour and one almost 'free' optimisation.

In terms of coding:

- `#include <sstream>`, `#include <vector>`, `#include <algorithm>` are not required. I can't see any stand-alone algorithms called. There are calls to member function equivalents, e.g. `m.lower_bound`, instead of `std::lower_bound`, but they're covered by the other headers.
- `#include <iterator>` should be added for `std::istreambuf_iterator`.
- `#include <string>` should be added for `std::string`.
- `argc` must be checked for the correct number of arguments and usage printed if no input file is specified.
- `ifs` must be checked for errors after it is created (and the file opened and read) – the file may not exist, may not be readable etc..
- It's a question of style, but I prefer explicitly stating the expected value in the declaration of `bool inword;` rather than using the default value.
- I'm not a fan of mixing `auto *start = ss.data();` and the `&ch` in the `range-for` loop – I know it'll work, but technically a `range-for` loop gets values by calling `.begin()` and `.end()` member functions, if they both exist, or `std::begin(..)` and `std::end(..)` otherwise. Yes, these all return type `CharT *`, so you can do the subtraction to supply the value for the second argument of the `std::string_view` ctor, but that's co-incidental. I'd consider using a for loop which isn't a `range-for` loop and initialise the start variable from `std::begin(..)` and go from there.
- The `auto & ch : ss` should be `const auto & ch : ss` as we're not going to change the contents of `ss`. If this change is made `auto *start = ss.data();` should also become `const auto *start = ss.data();`.
- Consider using `std::isalpha` instead of the comparisons and boolean logic setting the `letter` variable. There's minimal performance benefit to the logic and it's less expressive. If the logic must stay then put brackets around the `<=` clauses, as well as the `&&` grouping. Although the precedence is correct in the program the brackets make the expectation explicit.
- Prefer `for (const auto &entry : words)` over `for (auto &entry : words)`. We're not changing the value.
- Prefer `for (const auto &entry : m) {` over `for (auto &entry : m) {`. We're not changing the value.
- Consider not re-using the variable name `entry` as it can be a little confusing.

In terms of correctness:

- If the file ends on a word (and no non-alpha characters follow up until EOF) then that last word isn't counted. The solution is to either append a non-alpha character to `ss` after it is read, or put this code right after the word search loop:

```
if(inword) ++words[{ start,
  &ss[ss.size()] - start }];
```

  Personally, I'd implement the non-range-for loop described earlier in the `ss.data()` comment and incorporate an end-of-loop correction into that somehow.

- The `std::map` section of the code doesn't necessarily find the top ten words. Since we're using a `std::map` (and not a `std::multimap`) then the key has to be unique – so if two words have the same frequency only the first word is included in the map as later insert calls do not overwrite it. Which word is the first word is difficult to say as it'll be the first one from the `range-for` traversal of the **words** `std::unordered_map` – and `std::unordered_map` does not maintain an order to the elements. In fact it'll be a function of insertion order, the data structure used in `std::unordered_map` and the bucket count.

  There is a solution to this through the use of `std::multimap` instead – in fact it could almost be a drop-in replacement. However, because of the way the map is trimmed with `if (m.size() > 10)` if we have nine ties for first place, and three for second place, only one of the second place entries will be listed (the first one in alphabetical order). Similarly, if there is a 3-way tie for 1st, 3-way for 2nd, 3-way for 3rd and 3-way for 4th, only one of the 4th place values will be printed.

- In the `std::map` section the test `if (it != m.begin() || m.empty()) {` is wrong – it should be `if (it != m.begin() || (m.size()<10)) {`, otherwise under some conditions words with frequencies that don't go straight to the top of the leader-board will get lost.

  To print the words of the top ten highest frequencies (printing all ties) we could use a map of vectors:

```
std::map<size_t,
std::vector<std::string_view> > m;
for (const auto &entry : words) {
  auto it = m.lower_bound(entry.second);
  if (it != m.begin() || (m.size()<10)) {
    if(it->first==entry.second) {
      it->second.push_back(entry.first);
    } else {
     m.insert(it, { entry.second,
       { {entry.first} } } );
     if (m.size() > 10) {
       m.erase(m.begin());
      }
    }
  }
}
for (const auto &entry : m) {
  std::cout << entry.first << ":";
  for(const auto & w : entry.second)
    std::cout << ' ' << w;
  std::cout << '\n';
}
```

In terms of behaviour:

- The word matching is case sensitive. The matching can be made case-insensitive by converting all characters read to lower case before they're put into `ss` – with an iterator decorator. Alternatively, in the instantiation of `std::unordered_map` the `KeyEqual` and

`Hash` type template arguments (the third and fourth template arguments) could be replaced with types that ignore case.

- Word hyphenation is ignored. Should every word of "state-of-the-art' be counted separately?
- Apostrophes are also ignored. "won't" and "won" should not both increment the "won" count in my opinion. But if apostrophes are considered part of a word then "people's" and "people" wouldn't both increment people, and perhaps they should.
- We're printing (or at least trying to) the top ten most frequent words. Shouldn't the ten be configurable at runtime?

In terms of performance:

The 'free' optimisation is to use `istream::read` to read into a buffer, rather than use `std::istreambuf_iterator` instances to construct `ss`. Specifically, the file size must be found first, and a string of that size constructed, then the contents of the dummy string are overwritten with the file contents:

```
size_t getFileSize(std::ifstream & f) {
  auto prev_pos = f.tellg();
  f.seekg(0, std::ios::end);
  auto file_size = f.tellg();
  f.seekg(prev_pos);
  return file_size;
}
void usage(const char * execname) {
  std::cerr << execname << " input_file_name"
    << '\n';
}
int main(..) {
  const char * execname = argv[0];
  if(argc!=2) { usage(execname); return 1; }
  const char * infilename = argv[1];
  std::ifstream ifs{argv[1]};
  if(!ifs) {
    std::cerr << "Could not open file "
    << infilename << '\n'; return 2;
  }
  size_t filesize = getFileSize(ifs);
  std::string ss(filesize, ' ');
  ifs.read(ss.data(), filesize);
  auto * start = ss.data();
```

This is significantly faster than the previous approach to reading the file as the runtime is *probably* reading in chunks from the file – the exact way is implementation independent – it may be standard block reads or it could be that the file is mmaped. But we're telling the runtime that we're reading sequentially and in large chunks so the OS can read ahead and have the next data ready for the program.

We're also avoiding the string having to resize as data is read as we've provided the size in the constructor. Further, some OSes, such as Linux, don't actually allocate memory pages until they are touched – by initialising `ss` to `filesize` many space characters we're sure the memory is allocated and available.

Running this on a file which consists of ten copies of War and Peace (a 42MB file with 5.7 million words) the execution time is 74% of the original implementation. It's actually less than that, but once the file resides in the OS's filesystem cache the original program sees significant performance benefits to repeated executions.

In terms of design:

While the loading of the whole file into memory is doable on today's computers I'd suggest reading in a chunk of text at a time, extracting the words, storing only unique words (and only once), updating the tally, and getting the next chunk. Drop `ss` altogether and extract words from the chunks into `std::string` and store those directly in the `std::unordered_map`.

This is for two reasons – firstly, memory scalability (and a measurable performance improvement due to a lower cache load), and secondly to make the code more flexible by allowing incremental processing. The data may be coming from a network stream, and we won't know all the input data at the start, but we may want to know the ten most common words so far. Storing only one copy of every word means we're looking at most around 80000 words (a large English vocabulary size) held in memory, but never more than if we were storing the entire file in memory. In some cases, for example if we were looking for the ten most traded stocks in a ticker stream, we can know the entire vocabulary ahead of time.

A back-of-the-laptop implementation (using `std::string` instead of `std::string_stream` in the unique-word implementation, henceforth locally known as the store-only-unique-words-and-read-in-blocks or SOUWARIB version) had an execution time of between 70% of the original program, or almost as low as 50%, depending on the file size.

With regards to performance the elephant in the room is the STL implementation, I believe. CLang, GCC and Visual Studio STL implementations can vary internally quite a lot.

Doing some microbenchmarking on the SOUWARIB version, the bottleneck (for GCC 7.3.0 and libstdc++ 6.3.0) is by far the lookups in `std::unordered_map`. [1] has an excellent discussion of the different implementations each compiler uses for unordered associative maps.

The thing about the off-the-shelf implementations is that they have to be flexible and behave well enough in most circumstances. But we have a smaller problem space – for example, we have an upper limit to the number of unique words (ie the size of the OED, or thereabouts, or the number of symbols on a stock exchange), so we can allocate a fixed number of buckets and forgo the re-sizing and re-bucketing, and we can skip the NVI pattern [2] that an STL implementation might use to decrease compile times. We also know that we won't be removing any elements from our map, and so on..

Now for the parental guidance bit – this code is highly experimental, and somewhat of a hack, but it's a proof of concept. This is a drop in replacement for `std::unordered_map`, but only insofar as the features that are used in the code for this critique – it's missing most of the features of a general purpose implementation (and I've also edited out a few things for brevity).

```
#include <functional>  // std::hash
#include <vector>
#include <utility>     // std::pair
#include <iterator>    // std::next
#include <boost/iterator/transform_iterator.hpp>
#include <boost/iterator/filter_iterator.hpp>
template <typename KeyT, typename T,
  class Hash = std::hash<KeyT>,
  class KeyEqual = std::equal_to<KeyT> >
class my_hashmap {
protected:
  typedef std::pair<const KeyT, T>
    value_type;
  typedef std::size_t hash_type;
  class nodeT {
  protected:
    alignas(alignof(value_type))
    uint8_t data [sizeof(value_type)];
    hash_type key_hash;
    //MSB is set flag, rest is next offset
    size_t next_and_set = 0;
    constexpr static
    decltype(next_and_set) SET_MASK =
      (1ULL << ((sizeof(next_and_set)<<3)-1));
    constexpr static
    decltype(next_and_set) NEXT_MASK =
      ~SET_MASK;
    void set(bool newval) noexcept {
    next_and_set = (next_and_set & NEXT_MASK)
      | (newval?SET_MASK:0); }
  public:
    bool is_set() const noexcept {
```

```
    return next_and_set & SET_MASK; }
  bool isSameKey( const KeyT & other,
    const hash_type & other_hash ) const {
      return ( is_set() &&
        (key_hash==other_hash) &&
        KeyEqual()(get_cvalue().first,
          other) );
    }
    value_type & get_value() noexcept {
      return *reinterpret_cast<value_type*>
       (&data);
    }
    const value_type & get_cvalue() const
      noexcept {
        return *reinterpret_cast<const
          value_type*>(&data);
    }
    size_t get_next_index() const noexcept {
      return next_and_set & NEXT_MASK; }
    void set_next_index(size_t next)
      noexcept {
      next_and_set = (next_and_set&SET_MASK)
        | (next & NEXT_MASK);
    }
    nodeT() = default;
    nodeT(const KeyT & key,
      const hash_type new_hash,
      const T & val): key_hash(new_hash) {
      new (&data) value_type{key,val};
      set(true);
    }
    nodeT(const nodeT & o)
    : key_hash(o.key_hash),
      next_and_set(o.next_and_set) {
      if(o.is_set()) new (&data)
        value_type(o.get_cvalue());
    }
    // assign (copy|move) op, move ctor
    //  omitted for brevity
    ~nodeT() {
      if(is_set())
        reinterpret_cast<value_type*>(&data)
          ->~value_type(); }
  };
  const size_t n_buckets;
  std::vector < nodeT > storage;
  typedef typename decltype(storage)::
    iterator storage_iter;
public:
  my_hashmap( size_t log2_num_buckets )
  : n_buckets(1<<log2_num_buckets),
    storage(n_buckets) { }
  T & operator[] ( const KeyT & key ) {
    // no exception safety
    hash_type keyhash = Hash()(key);
    size_t bucket_index =
      keyhash & (n_buckets-1);
    size_t last_index = bucket_index;
    nodeT * node = & storage[bucket_index];
    if ( node->is_set() ) {
      bool found = false;
      while ( (!(found =
        node->isSameKey(key, keyhash))) &&
        node->get_next_index() )
        node = & storage[(last_index =
          node->get_next_index())];
      if(!found) {
        storage.emplace_back(key, keyhash,
          T{});
        node = & storage[last_index];
        node->set_next_index(storage.size()
          -1);
        node =
          &storage[node->get_next_index()];
      }
    } else {  // bucket not set yet
      node->~nodeT();
      new (node) nodeT(key, keyhash, T{});
    }
    return node->get_value().second;
  }
  constexpr static auto get_value_only =
    [](nodeT & n) -> value_type & {
      return n.get_value(); };
  constexpr static auto is_set_pred =
    [](nodeT & n) -> bool {
      return n.is_set(); };
  typedef boost::filter_iterator <
    decltype(is_set_pred), storage_iter >
    filtered_it_t;
  typedef boost::transform_iterator <
    decltype(get_value_only), filtered_it_t>
    value_iterator;
  value_iterator begin() {
    return value_iterator( filtered_it_t(
      is_set_pred, std::begin(storage),
      std::end(storage)), get_value_only  );
    }
  value_iterator end() {
    return value_iterator( filtered_it_t(
      is_set_pred, std::end(storage),
      std::end(storage)), get_value_only  );
  }
};
```

There's much wrong with this – from the lack of exception safety, risk of memory leaks, to the use of a power of two for the number of buckets (often number of buckets is a prime to limit bucket collisions). It really is just a back-of-the-laptop proof-of-concept. The point is that by dropping this in to SOUWARIB, the execution time is now around 70% of the **SOUWARIB** with **`std::unordered_map`** and around 50% of the original program.

With some help from Hardware Performance Counters [3] (link is for Intel, but AMD and ARM have equivalents) it can be shown that the performance jump in SOUWARIB+my_hashmap over SOUWARIB+**`std::unordered_map`** is due to significantly less memory accesses. Microbenchmarking with [4] shows that the cache hit ratio remains about the same in the word extraction loop but the number of accesses approximately halve.

For those not wanting to re-implement **`unordered_map`** or other STL functionality, have a look at EASTL [5] as there is some mildly non-standard, but otherwise very fast STL equivalent functionality in there.

References

[1] http://bannalia.blogspot.co.uk/2013/10/implementation-of-c-unordered.html
[2] https://en.wikipedia.org/wiki/Non-virtual_interface_pattern
[3] https://software.intel.com/en-us/articles/intel-performance-counter-monitor
[4] https://github.com/jasonspencer/CPP_LPE_wrap
[5] https://github.com/electronicarts/EASTL

Commentary

There's a lot in this, seemingly quite simple, critique. The first design issue is that of choosing to use **`string_view`**. For those not familiar with this feature, new in C++17, it is a non-owning view over a sequence of characters (it's loosely a wrapper for pointer + length). The advantage, from a performance point of view, is that there is no need to copy the data going to make up each word but it can be used in place. However, this has two disadvantages here. The first is that the original data must not be disposed of before the last use of any of the **`string_view`**s referring to

it. In this case it means we must leave the whole source file in memory as we don't know which portions of it are in use by the `unordered_map`. The choice is predicated on the decision that the program can only deal with data that can be read into memory in one go.

The second disadvantage here is that the keys in the map refer to memory with little cache locality and this has a negative effect on performance with most modern hardware. as Jason demonstrates towards the end of his critique reduction in memory access cost is often the most significant change that you can do to improve the performance of a program limited by compute time.

The subsidiary design problems mostly concern boundary conditions:

- what should be done about punctuation and capitalisation
- in some languages you may also have to consider accents: Paul in particular hit this problem
- what to do with a word 'in progress' at the end of the file
- what to do with 'ties' for counts

The entries between them covered most of these issues – for example all suggested `std::isalpha()` rather than hand-crafted comparisons.

Pal expressed a concern over using a temporary for the call to `insert`. Here for example `emplace_hint` could be used instead (as Scott Meyers writes in *Effective Modern C++*: 'Consider emplacement instead of insertion').

As Jason states, the STL containers are good general purpose containers; the implementors work hard to ensure that they deliver the best overall performance for a range of possible use cases. In any specific program, especially where you the programmer may have additional knowledge about the data sets being processed, it may well be possible to craft a tailored collection or algorithm that out performs the one in the standard library.

However, it is also likely that any replacement you provide will contain its own bugs and it's also important to verify that your proposed solution is in fact faster than the standard library component as these have in general had a lot of design put into them over many years!

I personally find myself a little torn over the style of using `{}` for initialising variables. One advantage is consistency: you can use the same syntax for a variety of different data types without having to read and parse the various possible initialisation values. One disadvantage is that it relies on the reader knowing what the value is for each type – `bool inword{false}` makes this explicit. Alternatively, using the syntax `bool inword = false` has the advantage that it is more like the syntax that would be used in some other programming languages. I suspect disagreements about the best style will continue as it does not seem that there is an obvious reason to prefer one style.

Lack of error handling is a continual problem with the code critique challenges – as it is in quite a lot of other code. Anything that relies on external inputs, such as the presence of command line arguments and the existence of files, should have some sort of check as these are the sort of things that are very likely to go wrong and can also waste a surprisingly large amount of time in resolving.

The Winner of CC 110

The four critiques all did a good job of identifying a number of things to improve in the submitted code. James suggestion for using regex to do the parsing does help to produce more robust and flexible code (although unfortunately this seems to come at a price in terms of performance.) Paul described some of the problems with handling French (other languages may have similar issues) and suggested switching to a whitespace/punctuation based approach.

Pal's intuition that there might be a standard algorithm already written for use in the final stage was sound; he found this in `partial_sort_copy()`. Few people know the whole set of standard algorithms so it is well worth having a quick look when you are doing something relatively straightforward to avoid having to re-invent the wheel!

However, I think that overall Jason did the best job of identifying both some small nits but also the larger design issues and pointing towards the sorts of things that need considering when you've 'tried to make it pretty fast'. It will depend on the specific use case whether the performance needs in this case would justify the extra development cost of writing a bespoke hash map, and one of the problems with the code critique format – as Pal said – is that there is no mechanism for asking those sorts of questions. So, congratulations to Jason and thanks to the other entrants.

I note that all of these entrants have submitted several times before; can I encourage those of you who haven't quite got round to sending in an entry to give it a go?

Code Critique 111

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

I'm playing around with simple increasing sequences of small numbers and seeing what various different generators produce. I had problems that sometimes the result was wrong – for example if I produce a number that overflows the integer size I'm using – and so I added a postcondition check that the output sequence is increasing. However, I've found the check doesn't always 'fire' – it works with gcc with or without optimisation but fails with MSVC unless I turn on optimisation. I've produced a really simple example that still shows the same problem, can you help me see what's wrong?

Expected output, showing the postcondition firing:

```
1,2,3,4,5,6,7,8,9,10,
1,3,6,10,15,21,28,36,45,55,
1,1,2,3,5,8,13,21,34,55,
1,5,14,30,55,91,140,204,285,385,
Postcondition failed!
1,5,14,30,55,91,-116,-52,29,-127,
```

Can you help (and perhaps identify one or two other issues en route?) The code is in Listing 2.

Listing 2

```
#include <algorithm>
#include <functional>
#include <iostream>
#include <iterator>
#include <vector>
class postcondition
{
public:
  postcondition(std::function<bool()> check)
  : check_(check) {}
  ~postcondition()
  {
    if (!check_())
      std::cerr << "Postcondition failed!\n";
  }
private:
  std::function<bool()> check_;
};
template<typename T>
std::vector<T> get_values(int n,
  std::function<T(T)> generator)
{
  std::vector<T> v;
  auto is_increasing = [&v]() {
    return is_sorted(v.begin(), v.end()); };
  postcondition _(is_increasing);
  T j = 0;
  for (int i = 0; i != n; ++i)
  {
    j = generator(j);
    v.push_back(j);
  }
  return v;
}
```

Local ACCU Meetups

Frances Buontempo reports from meetups in Bristol and London.

Bristol

28th February: 'Simply the best: optimizing with an evolutionary computing framework'

Chris Simons and I gave a practical workshop in using evolutionary algorithms in Java. I travelled there with Steve Love through the snow and back through more snow. The snow meant only a small handful of people made it.

We started with a brief introduction to evolutionary algorithms, including an overview of the many freely available frameworks for optimization with evolutionary computing. Next, we used the open source Java Class Library for Evolutionary Computing (JCLEC) (http://jclec.sourceforge.net). The attendees got a chance to program with the framework to address three optimization challenges:

1. 'OneMax', a 'hello world'-type example for evolutionary algorithms,
2. 'How to program you way out of a paper bag' by firing cannon balls, and
3. The 'travelling salesman problem' (or TSP), wherein you must visit everywhere in a list of places, returning where you began and trying to find the shortest route. TSP is like a kata for Machine Learning. TSP-type problems tend to crop up quite a lot.

We didn't have enough time to go through the last two in detail, but this was useful practice for the session we gave at the ACCU conference. Chris, and a colleague Aurora Ramirez, previously wrote about the framework for *Overload* if you want to try it out. [1]

London

12th February: '.Net Code Craft'

Jason Gorman, Managing Director of Codemanship [2] ran this session. We discussed the five factors that make code more difficult and expensive to change, and explored how we can write code that delivers value today, and leave the door open to delivering more value tomorrow.

We looked at:

1. Automated tests, and how they help us make changes safely and economically
2. Readability (and why comments are a cop-out)
3. Complexity and the ways it hurts us
4. Duplication (and how removing it can reveal better designs)
5. Dependencies and how to minimise the 'Ripple Effect'

The presentation included demonstrations using Visual Studio, ReSharper & NUnit.

We had a group of something like 20 people, including several new faces. Having meetups on a variety of different topics pulls in a variety of

FRANCES BUONTEMPO

Frances has a PhD in machine learning and data mining. She has been a programmer since the 90s, and learnt to program by reading the manual for her Dad's BBC model B machine. She can be contacted at frances.buontempo@gmail.com

Code Critique Competition (continued)

Listing 2 (cont'd)

```
using out = std::ostream_iterator<int>;
template <typename T>
void sequence()
{
  auto const & result = get_values<T>(10,
    [](T i) { return i + 1; });
  std::copy(result.begin(), result.end(),
    out(std::cout, ","));
  std::cout << std::endl;
}
template <typename T>
void triangular()
{
  T i{};
  auto const & result = get_values<T>(10,
    [&i](T j) { return j + ++i; });
  std::copy(result.begin(), result.end(),
    out (std::cout, ","));
  std::cout << std::endl;
}
template <typename T>
void fibonacci()
{
```

Listing 2 (cont'd)

```
  std::copy(result.begin(), result.end(),
    out (std::cout, ","));
  std::cout << std::endl;
}
template <typename T>
void sum_squares()
{
  T i{1};
  auto const & result = get_values<T>(10,
    [&i](T j) { return j + i * i++; });
  std::copy(result.begin(), result.end(),
    out (std::cout, ","));
  std::cout << std::endl;
}
int main()
{
  sequence<int>();
  triangular<int>();
  fibonacci<int>();
  sum_squares<int>();
  sum_squares<char>(); // overflow expected
}
```

ACCU Local Group

Jason Spencer reports on a recent visit to ACCU Oxford.

I started regularly attending ACCU London meetings almost two years ago. At the time, I just missed Jason McGuiness's talk 'Knuth, Amdahl: I spurn thee!' [1] but luckily I recently had a chance to attend an updated talk [2] on the 27th of March 2018 at the ACCU Oxford chapter.

To a packed room, Jason told us about the ins and outs of CPUs, compilers and how to write C++ code to get it executed as fast as possible. The topic is a broad one, and Jason's treatment of it was very thorough.

Jason started by describing the problem of low latency systems and the issues in an HFT (High Frequency Trading) context – the system isn't safe-critical, for example, but speed is paramount, even the slightest improvement can make all the difference. Jason covered the environment in which these systems operate: multiple cores, discussing system buses, operating systems, compilers and libraries, and the most important part being the internal units of a modern CPU with a super-scalar architecture, deep pipelines, and SIMD units.

We were then led down the rabbit hole of assembly code generation and the behaviour across compilers and even the chaos across different versions of the same compiler. I won't go into the details, nor parrot Jason's examples here, but I strongly recommend viewing the slides [3] as they are very well explained.

Jason has also gone to great pains to precisely microbenchmark the performance of different compilers for the same code, and elaborated on why in some cases there were such significant changes in performance.

Later topics covered included static branch prediction, switch statements and the use of compile-time techniques to direct the compiler to generate optimal code (for example using C++ templates to create a faster memcpy). All with the support and justification of extensive micro-benchmarking and disassembled code.

Of course this was all with the usual flair and entertaining style of Jason's presentations. As already mentioned Jason has kindly provided the slides from his talk at [3]. Alas, there was no video recording in Oxford, nor of his 2016 talk at ACCU London. Jason also makes some related slides available here [4].

JASON SPENCER

Jason is a software engineer interested in high performance computing, distributed processing, Complexity theory, and C++. He has degrees in Electronic Engineering and Computer Science, and a PhD in Telecommunications. He can be contacted at contact@jasonspencer.org

Local ACCU Meetups (continued)

different people, but we don't always manage to persuade everyone to join us for a drink afterwards, so didn't get a chance to get to know the new faces. Jason has spoken to us before a long while ago and always presents well.

29 March: 'The fantastic four coding patterns of Continuous Delivery (CD)', followed by 'An introduction to data science'

The first speaker, Luca Minudel, is a Lean-Agile Coach and Trainer, and founder and CEO at SmHarter.com. He contributed to the adoption of lean and agile practices by Ferrari's F1 racing team. When he joined Scoured Ferrari F1 racing team in 2006, he was asked to increase the speed of software development while at the same time increasing the reliability and reducing the number of bugs. That sounded like a paradoxical puzzle to him. And he wondered why didn't he know the solution already, given that he was hired as an expert. This challenge, in a high-pressure fast-paced environment, led him to the find four CD coding patterns, two that are known nowadays, and two that are new. He told us the story of this discovery and the learning, and introduced the four patterns that enabled his team to increase the speed of software development without the need to trade speed for quality or safety.

He talked through how to cope with a version control system that doesn't support branching and showed how to make safe steps, allowing roll-back in case of trouble. This was a practice run through for his ACCU conference talk, and this is now on the ACCU YouTube channel [3]. Several other talks are up there too, so search for '[ACCU2018]'.

The second talk was an introduction to data science/machine learning using python. Shagun Khare gave this session. She finished her Masters in Data Science/Machine Learning at City, University of London recently, so I invited her to do this. Having two talks in one evening was ambitious, but it worked out. She used a Jupyter notebook to work through a handful of algorithms.

This introduction to machine learning considered where it sits in Data Science process, and covered supervised and unsupervised learning. Shagun walked through steps in machine learning:

- splitting data into test and training,
- fitting a model,
- hyperparameter optimisation,
- predict from the model,
- evaluating the predictions.

She gave a code demo using a small dataset, using Anaconda, allowing the audience to play along with the notebook. This is on Github if you want to try it out [4]. The talk covered several common terms and some classification algorithms using the iris dataset, which I used in Visualisation of Multidimensional Data *CVu*, 29(6):3-5, January 2018. It's a relatively small dataset of three types of iris flowers, which is often used to show case machine learning algorithms.

I'd love to see other people write up what's happened at their local groups, or start one if you aren't near one already.

References

[1] Aurora Ramirez and Chris Simons (2017) 'Evolutionary Computing Frameworks for Optimisation': *Overload* 142 December 2017. https://accu.org/index.php/journals/2444

[2] Codemanship: http://www.codemanship.com

[3] Luca Minundel (2018) https://www.youtube.com/watch?v=rw9I39nUkXI

[4] https://github.com/Shagun8ProductMadness/IntroToMachineLearning

View from the Chair

Bob Schmidt
chair@accu.org

The day-to-day operations of ACCU continue to tick along, thanks to the generous donations of time and effort by all of ACCU's committee members and volunteers.

After five years of declining membership numbers, ACCU finished 2017 with a small increase in membership (673) compared to the end of 2016 (649). The 2017 ACCU Conference also saw an increase in attendance over 2016. (It is possible that the two trends are related, with people joining ACCU in order to take advantage of the conference discount.) While these are positive trends, it is important to note that our total membership of 673 at the end of 2017 still is approximately 31% lower than the 981 members ACCU had in 2007 (the last year for which I could find numbers).

ACCU had other successes in the past year:

- The conference web site continues to be improved, thanks to our Conference Chair, Russel Winder, and his collaborators.
- ACCU's financial situation remains strong, with ACCU finishing 2017 with a surplus for the fiscal year. The 2017 surplus was 19.4% greater than 2016's surplus.
- Local group ACCU Cambridge was started up, thanks to the efforts of Andy Saul.
- *CVu* and *Overload* continue to be high-quality publications, thanks to editors Steve Love and Fran Buontempo, our reviewers, and our member and non-member contributors.
- We published our new, generalized Code of Conduct, outlining our expectations for people attending any event held under the ACCU banner. (The ACCU Conference will continue to use its more detailed Code of Conduct.)
- Our local group affiliates have continued to experience strong membership growth. (I would warn against reading too much into those numbers, however. A Meetup group has no barrier to entry, and raw numbers don't necessarily reflect the actual number of people participating in the group. Also, the large increase in local group numbers has not resulted in a similar increase in paid ACCU membership.)
- ACCU awarded an ISDF grant to Mr. Walter Brown for his contributions to WG21, the C++ standards committee.

We still have challenges:

- We continue to find it difficult to recruit volunteers for committee positions. The Web Editor, Publicity, Study Groups, Social Media, and Book Reviews positions remain vacant. The position of Secretary will become vacant as of the AGM, unless someone steps forward and runs during the meeting. Our current treasurer, Rob Pauer, would like to retire, but has graciously agreed to remain in the role until someone volunteers to replace him.
- We are considering moving our web site off of its current platform, Xaraya, and onto a more modern, supported platform. For most things Xaraya is satisfactory, but it hasn't been updated since October 2015, and its facilities for preventing bot accounts from being created is lacking. (We currently get 100 bot sign-ups per day, with most coming from just three sources.) This will be a long-term project for the committee.

I would like to take this opportunity to thank Malcolm Noyes for his service on the committee. Malcolm has served as Secretary for four years, and has decided to retire as of this year's AGM. Malcolm tried to retire two years ago, but agreed to remain as acting Secretary until the special election of 2016, and ran for, and won, the position again last year. He will be missed.

Finally, a big thank you to all of ACCU's volunteers: committee members; magazine editors, writers, and reviewers; local group coordinators; and conference committee members.

Local groups

Emyr Williams has notified the committee that he will be standing down as the local group coordinator for Bristol and Bath. (He intends to remain in his position of Standards Officer.) Please join me in thanking Emyr for his work on the local groups.

Call for volunteers

Would you like to be thanked, in this very magazine, by yours truly? Volunteer for any one of ACCU's open positions, and see your name somewhere in this column in a future issue of *CVu*.

- Secretary
- Web Editor
- Book Reviews
- Publicity
- Study Groups
- Social Media

Portions of this View appeared in the Annual General Meeting Information Pack as part of the Chair's Report.

ACCU Local Group (continued)

Compared to ACCU London, which is most often hosted at Code Node/ Skills Matter near Moorgate, this ACCU Oxford meetup (and most of their meetups) was hosted in the function room at St. Aldate's Tavern in the centre of Oxford. A very nice and cosy venue with all the benefits of a traditional Victorian pub.

It was also very nice to meet other ACCU members that are based outside London, and to put names to faces on the mailing lists and publications. Of course the ACCU conference in April is another great such opportunity.

For someone based in London, the trip up to Oxford, as I'm sure most will know, isn't a tough one – a direct train from Paddington or Marylebone – a journey of about 57 minutes to 1 hr and 15 minutes depending on the time of day and the train operator/route (as well as the alignment of the stars, a butterfly in China and snow taxonomy). If you can get out of work in London a little before 17.30 (often wishful thinking, I know) then, although a little tight, it's possible to get to the tavern by 19.00, as in fact Jason did, coming from Canary Wharf, in time for a 19.05 start.

I do recommend the occasional (or even regular) trip to ACCU Oxford – the crowd are slightly different: in London it's more people working in finance, and some from gaming, database products, or a start-up; but in Oxford while there are some people working in finance, from my limited experience there is a more diverse group: biotech, scientists, researchers.

I look forward to making the trip again sometime soon.

References

[1] https://www.meetup.com/ACCULondon/events/229636136/
[2] https://www.meetup.com/ACCU-Oxford/events/245778124/
[3] https://www.slideshare.net/JasonMcGuiness/knuth-amdahl-i-spurn-thee-92110552
[4] https://www.slideshare.net/JasonMcGuiness/presentations

"The magazines"

The ACCU's *C Vu* and *Overload* magazines are published every two months, and contain relevant, high quality articles written by programmers for programmers.

"The conferences"

Our respected annual developers' conference is an excellent way to learn from the industry experts, and a great opportunity to meet other programmers who care about writing good code.

"The community"

The ACCU is a unique organisation, run by members for members. There are *many* ways to get involved. Active forums flow with programmer discussion. Mentored developers projects provide a place for you to learn new skills from other programmers.

"The online forums"

Our online forums provide an excellent place for discussion, to ask questions, and to meet like minded programmers. There are job posting forums, and special interest groups.

Members also have online access to the back issue library of ACCU magazines, through the ACCU web site.

ACCU | JOIN: IN

PROFESSIONALISM IN PROGRAMMING
WWW.ACCU.ORG

Invest in your skills. Improve your code. Share your knowledge.

Join a community of people who care about code. Join the ACCU.

Use our online registration form at **www.accu.org**.

Design: Pete Goodliffe