the magazine of the accu

www.accu.org

{cvu}

Volume 28 • Issue 1 • March 2016 • £3

Features

Sliding Windows Filters:
A Set-based Implementation
Omar Bashir

Groovy and Grails eXchange 2015
Ralph McArdell

Using Clara to Parse Command Lines in C++
Malcolm Noyes

Software Development Is...
Pete Goodliffe

An Open Question (or How I Learned
To Stop Worrying And Love Public Wi-Fi)
Vertices

Regulars

C++ Standards Report
Book Reviews
Code Critique

{cvu}
Volume 28 Issue 1
March 2016
ISSN 1354-3164
www.accu.org

Cross Pollinated

I don't think it's controversial to say that we, as computer programmers, work in a young industry. By most estimates, it all kicked off in the 1940s, mostly in response to the need for a way of reading encrypted messages faster than doing it with a pencil and paper. The theoretical basis and history of the first attempts at programmable computers is probably familiar to most of us, with Alan Turing publishing a paper about a 'Universal Turing Machine' in 1937, Tommy Flowers' Colossus clattering to life at Bletchley Park in 1943, all making way for IBM to stamp out most commercial competition in the 1950s.

Many illustrious names litter this history of programming. Charles Babbage's mechanical Difference Engine in the 1820s was a precursor to the Analytical Engine, which gives us the first programmer – the Countess Ada Lovelace. The design of the Engine is significant by itself, too, given the striking similarities to the architecture proposed by John von Neumann for general purpose computers in the 1940s, and used in most (all?) computers until the 1980s, and beyond. Countess Lovelace herself was depending on 9th century work by the Persian mathematician Muhammad ibn Musa al-Khwarizmi, who gives his name to the term 'algorithm'.

Despite being uncomfortably aware that the names I have already mentioned contain that of but one woman, I will press on with my story of another man. In 1866, the first single cable was laid across the Atlantic Ocean between the UK and the US. It marks, arguably, the beginning of the modern Internet. There was only one ship capable of carrying the enormous weight of cable required: the SS Great Eastern, the largest ship ever built (a record held until 1901), was designed by Isambard Kingdom Brunel. Computer programming itself is certainly a young industry, at a commercial level, but it depends a great deal on the progress made by some unlikely people. I would naturally love to hear about your favourites, at the usual address.

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

18 Standards Report
Jonathan Wakely reports from the latest C and C++ meetings.

19 Code Critique Competition
Competition 98 and the answer to 97.

REGULARS

23 Books
From the bookshelf.

24 ACCU Members Zone
Membership news.

24 Regional Meeting Report
A report from the January 2016 London meeting.

FEATURES

3 Sliding Window Filters: A Set-based Implementation
Omar Bashir presents an implementation using pre-sorted data to reduce CPU load.

7 An Open Question (or How I Learned to Stop Worrying and Love Public Wi-Fi
Vertices examines some of the dangers of using other people's networks.

10 Groovy and Grails eXchange 2015
Ralph McArdell reports on his conference experiences.

12 Fifteen Love
Baron M sets a new puzzle for students of curious conundrums.

13 Using Clara to Parse Command Lines in C++
Malcolm Noyes demonstrates how to get up and running.

15 Software Development Is...
Pete Goodliffe defines the art, science, craft (and child's play) of software development.

SUBMISSION DATES

C Vu 28.2 1st April 2016
C Vu 28.3: 1st June 2016

Overload 133: 1st May 2016
Overload 134: 1st July 2016

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

Sliding Window Filters : A Set-based Implementation

Omar Bashir presents an implementation using pre-sorted data to reduce CPU load.

Sliding window filters are commonly used for noise reduction and data smoothing in applications ranging from algorithmic trading in financial systems to signal processing in digital media systems. Efficient implementations of these filters optimise overall performance of systems in which they operate.

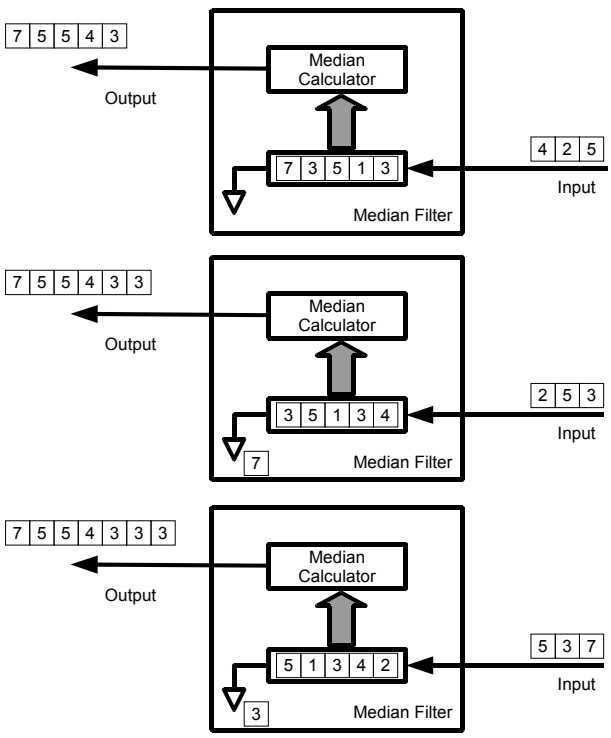
A set-based implementation of a sliding window filter is discussed which optimises the performance of the most common of sliding window filters, the median filter. This implementation can easily be extended to use measures other then median for filtering.

Motivation

A median filter operates on a window over the data stream. This window is implemented using a queue of a specified size. When the queue is full, an input causes the earliest value in the queue to be ejected and discarded. The output of the filter is the median of the values in the queue (see figure 1).

An obvious implementation of the Median Calculator in figure 1 sorts the data in the queue and then calculates the median from the sorted collection. An alternative and efficient implementation maintains a sorted collection which is updated with every new insert. If the queue is full, this update involves removing the earliest value from both the queue and the sorted collection and then adding the new value into both. Updating a sorted collection is generally a computationally less expensive operation than sorting the complete list (i.e., $\ln(n)$ vs. $n \ln(n)$). If the sorted collection is index based, calculation of median from it is a constant complexity operation. Figure 2 (overleaf) shows operation of a median filter using a sorted collection.

Figure 1



Sorted collections

Bashir [1] demonstrated the versatility of sorted collections of numerical data in the context of performance monitoring of computer networks. Analysis of network latencies and investigation of related incidents requires iterative derivation and analysis of several summaries at different levels of temporal granularities. Processing these statistical summaries from raw data in every iteration can be computationally intensive.

Bashir proposes preprocessing raw data to intermediate information at the lowest level of granularity. This intermediate information should have properties of richness, efficiency and mergeability. Richness is the ability to derive most statistical summaries from intermediate information without accessing the raw data. Efficiency requires calculation of statistical summaries to be equally or less computationally complex than deriving them from raw data. Finally mergeability is the ability to combine appropriate intermediate information at finer granularities to provide intermediate information at coarser granularities. These can then be processed efficiently to derive required summaries at those granularities.

Sorted numerical data exhibit all these properties. In the context of filtering based on measures of central tendency, richness and efficiency are desired properties while mergeability is less relevant as these filters generally operate at fixed granularities.

Using sorted numerical data also allows decomposing such filters into a preprocessor and a summary calculator. The preprocessor queues the input and maintains a sorted copy of the queued data. The summary calculator produces the desired measure of central tendency from the sorted collection. Thus, the output of the summary calculator is the filtered value. Decoupling the two allows developing filters based on different measures of central tendency using the same preprocessor implementation.

However, such preprocessors need to maintain a sorted copy of the window in addition to the window itself. This can be an issue for larger window sizes in applications that need to have a smaller memory footprint.

A sliding window preprocessor implementation

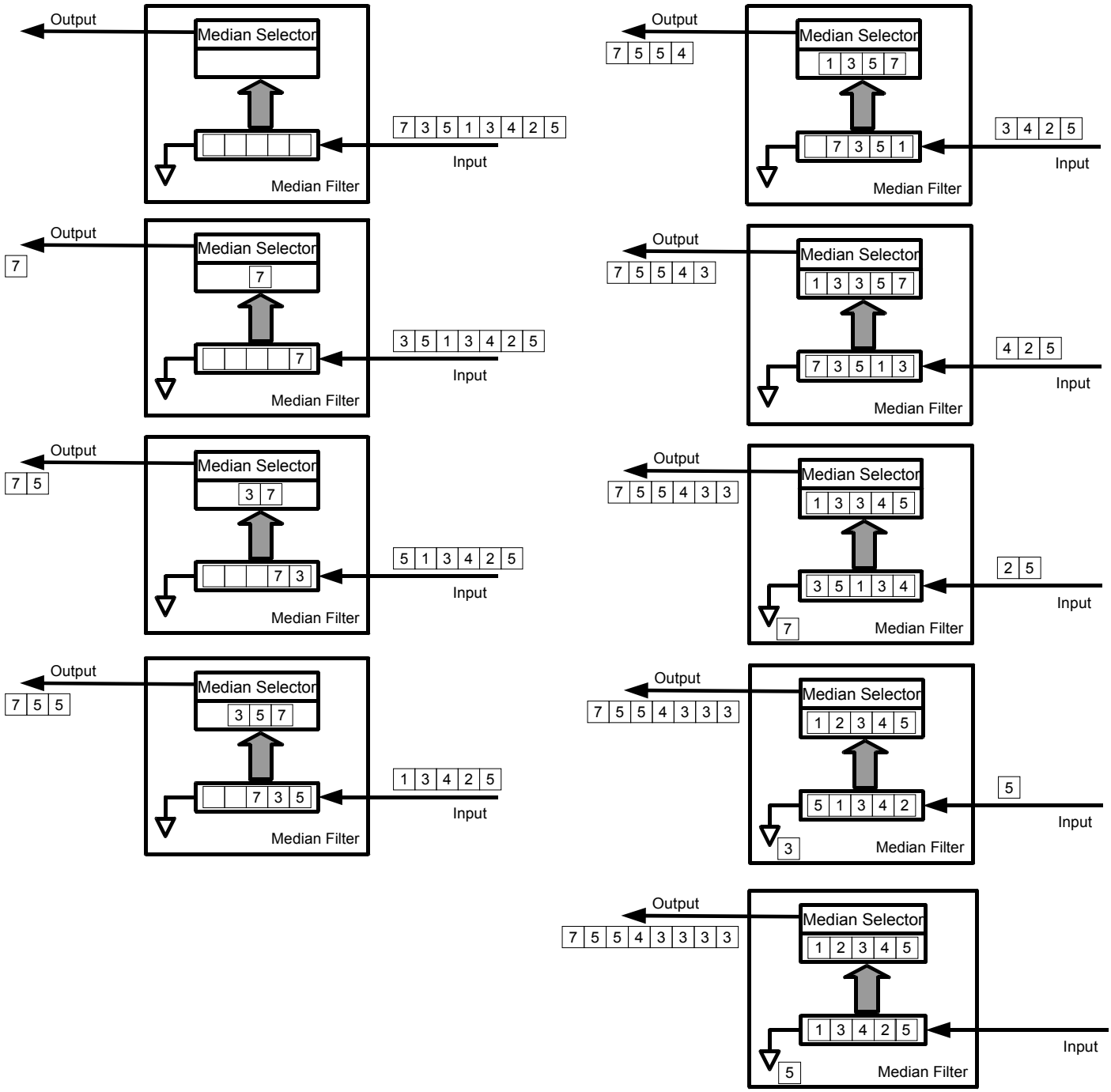
A Java implementation of a sliding window preprocessor is described here. This preprocessor uses an instance of the `ArrayBlockingQueue`, an implementation of the `Queue` interface, as the sliding window. A `TreeSet` instance is used as the sorted collection of elements in the sliding window.

A `TreeSet` does not allow recurring values while a sliding window algorithm accepts recurring values. To allow recurring values in a `TreeSet`, each value is encapsulated with a monotonically increasing integer that is incremented for every new value. `OrderedElement` class

OMAR BASHIR

Omar has developed software for domains ranging from defence and telecommunications to financial services. He is passionate about developing technology that is useful, usable and simple. He may be contacted at obashir@yahoo.com

Figure 2



(Listing 1) encapsulates the order of arrival with the arriving value. `OrderedElement`'s implementation of the `Comparable` interface compares the order of arrival for recurring values. This allows recurring

Listing 1

```
public class OrderedElement<T extends
Comparable<T>> implements
Comparable<OrderedElement<T>>{

  private Integer order;
  private T value;

  public OrderedElement(Integer order, T value) {
    this.order = order;
    this.value = value;
  }

  public Integer getOrder() {
    return order;
  }
```

Listing 1 (cont'd)

```
  public T getValue() {
    return value;
  }

  @Override
  public int compareTo(OrderedElement<T> o) {
    int valueComp = value.compareTo(o.value);
    return (valueComp != 0) ? valueComp
      : order.compareTo(o.order);
  }
  @Override
  public boolean equals(Object o) {
    return (null != o) &&
      (o instanceof OrderedElement<?>) &&
      ((this == o) ||
      (this.compareTo((OrderedElement<T>) o)
       == 0));
  }
 ...
}
```

values encapsulated in an `OrderedElement` instance to exist in a `TreeSet` in sorted order.

Listing 2 shows the preprocessor, `SetOrderedWindow`. An instance of this class uses `queue`, an `ArrayBlockingQueue` instance, and `sorter`, a `TreeSet` instance. `SetOrderedWindow` maintains an integer, `sequence`, which is incremented every time the `insert` method is called to insert a value into the window. If there is still capacity in the window, the input value is encapsulated with the sequence number in an instance of `OrderedElement` and inserted into both `queue` and `sorter`. If there is no capacity in the window, `SetOrderedWindow` removes the element at the head of the `queue` and also removes the same element from `sorter` before inserting the new element to both `queue` and `sorter` (Figure 3).

The `SetOrderedWindow` class contains two methods to retrieve the data its instance contains. The `getElements` method returns a `List` of all the values contained in `queue` in the order of arrival. `getElementsSorted` method returns the values contained in `sorter` in sorted order in an array which allows efficient index based access to the values. `getElementsSorted` is the most commonly used method for data access as the array it returns can be used efficiently to derive any measure of central tendency for filtering.

Listing 2

```
public class
    SetOrderedWindow<T extends Comparable<T>> {
  private
    ArrayBlockingQueue<OrderedElement<T>> queue;
  private Set<OrderedElement<T>> sorter;
  private int sequence;
  public SetOrderedWindow(int size) {
    this.sequence = 0;
    this.queue = new
     ArrayBlockingQueue<OrderedElement<T>>(size);
    this.sorter = new
     TreeSet<OrderedElement<T>>();
  }
  public int getCurrentSize() {
    return queue.size();
  }
  public int getCapacity() {
    return queue.size() +
      queue.remainingCapacity();
  }
  public void insert(T value)
      throws InterruptedException {
    OrderedElement<T> element =
      new OrderedElement<>(sequence, value);
    sequence++;
    if (queue.remainingCapacity() == 0) {
      OrderedElement<T> elementToRemove
        = queue.poll();
      sorter.remove(elementToRemove);
    }
    queue.put(element);
    sorter.add(element);
  }
  public ArrayList<T> getElementsSorted() {
    ArrayList<T> reply = new ArrayList<T>();
    for (OrderedElement<T> element : sorter) {
      reply.add(element.getValue());
    }
    return reply;
  }
  public List<T> getElements() {
    List<T> reply = new ArrayList<T>();
    for (OrderedElement<T> element : queue) {
      reply.add(element.getValue());
    }
    return reply;
  }
}
```

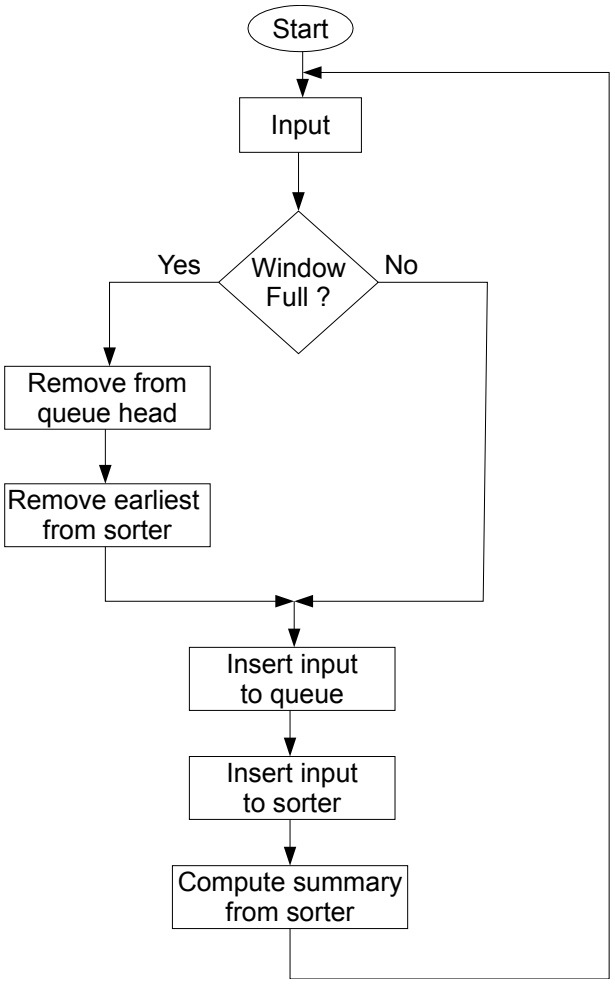


Figure 3

Performance measurements

The performance of the `SetOrderedWindow` class has been measured for different window sizes from 500 to 2500 elements.

The performance of `SetOrderedWindow` is compared with the performance of sorting the data in `queue` using an `ArrayList` for every summary query. These test applications were compiled using JDK 8 and executed over Lubuntu 14.04 on a 1.6 GHz Intel Atom N270 processor with 1 GB RAM.

Figure 4 compares the performance when the windows are being filled from the beginning and they are not yet full. This means the elements are only being inserted in the `queue` objects. In the `SetOrderedWindow`, these elements are also being inserted in `sorter` to keep the data in sorted order. For each window size, the time measured is the time required to fill the entire window and for each entry calculating the median of the data in the window.

Figure 5 compares the performance when the windows are already full. Inserting an element in the windows requires removing the element at the head of `queue` objects before inserting the new element at the tail of `queue` objects. In the `SetOrderedWindow`, the element at the `queue` head is also removed from `sorter` and the element being inserted at the tail of `queue` is also inserted into `sorter`. For each window size, the time measured is the time required to replace the contents of the entire window and for each new entry calculating the median of the data in the window.

These results show that the sliding window filter using a set based sorter performs better than the filter that sorts window data for every query. Moreover, relative performance of the set based sliding window filter improves for larger window sizes.

Most applications will have window sizes less than 100 elements. The performance difference between the two approaches may not be significant for an individual insert and summary calculation. But as these applications are long running, any additional performance impact reduces the overall throughput. However, a sliding window implementation using a set based sorter consumes more memory to maintain the data it needs to process. Depending upon the specifics of the implementation, it can be twice as much memory. As long as this overhead is acceptable, such an implementation can improve overall application throughput.

Conclusions

Sliding window filters summarise data captured in a window sliding over the input data stream and the summarised value at the output is the filtered value. These filters are used for noise reduction and data smoothing in applications from diverse domains such as financial computations, signal processing and automatic systems control.

A common sliding window filter is the median filter where the filter calculates median over the window and uses that as the filtered output. Conventional median filter implementations sort the data in the window for each insert. As sorting is computationally expensive, this can reduce throughput of host systems.

An alternative implementation discussed here uses a set to keep a copy of the data in the sliding window in sorted order. It is shown here that maintenance of this set is computationally less expensive than resorting the window as it slides over the input data. This, however, can have the cost of doubling the memory footprint of the filter because the filter needs to maintain the values in the window in arrival and in sorted order.

Maintaining sorted data in the filter allows calculating a range of summaries efficiently. Furthermore, actual summary calculation can be decoupled from maintaining the window in arrival and sorted orders inside a preprocessor. Thus, the same preprocessor can then be used with filters that use measures of central tendency other than the median.

While this article discusses implementing a 1-D sliding window filter, an adaptation can be used to implement 2-D sliding window filters. These can be used to optimise image processing applications. ■

References

[1] Bashir, O. (1998), Management and Processing of Network Performance Information, PhD Thesis, Loughborough University

Figure 4

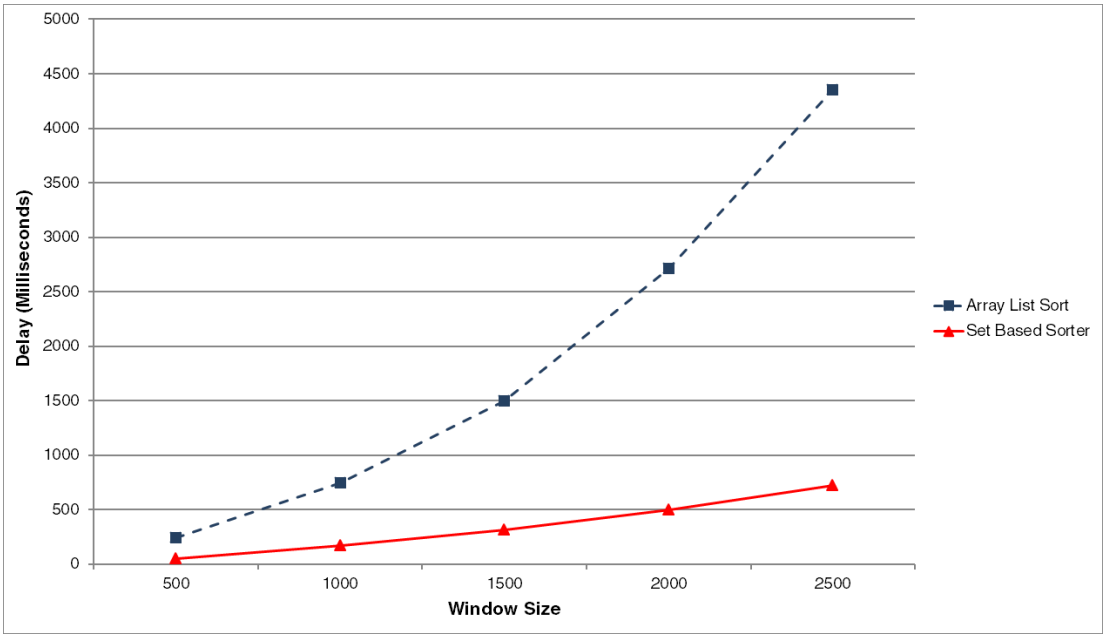
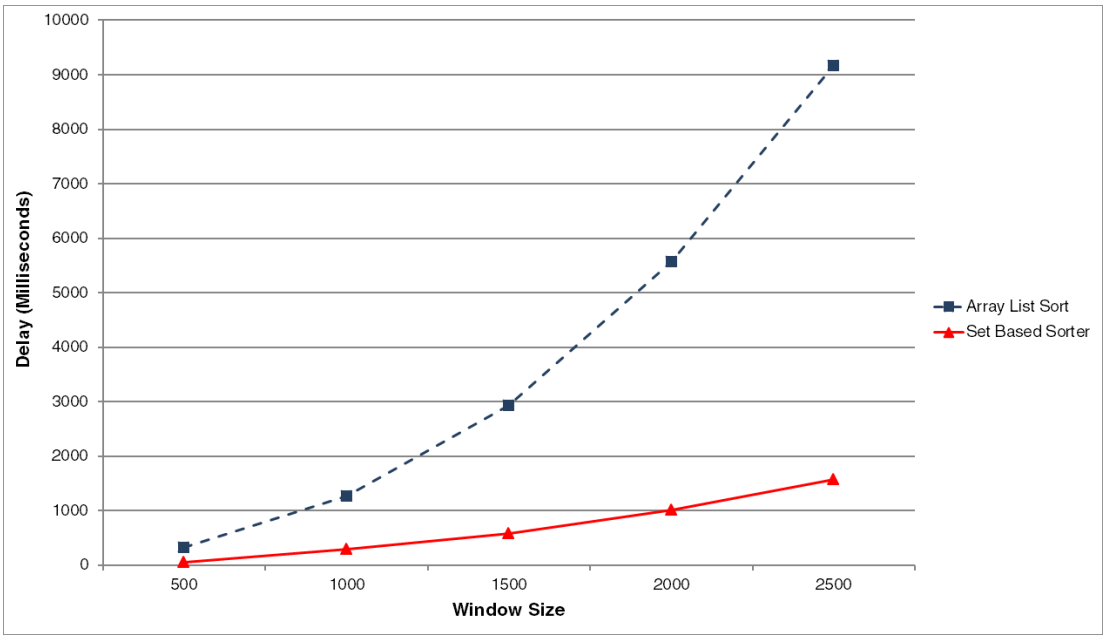


Figure 5



An Open Question (or How I Learned To Stop Worrying And Love Public Wi-Fi)

Vertices examines some of the dangers of using other people's networks.

A paranoid is someone who knows a little of what's going on.
~ William S. Burroughs

Just because you're paranoid doesn't mean they aren't after you.
~ origin unknown

The term 'computer virus' was coined over 30 years ago [1] to describe self-replicating computer programs, and since then, we've become very well acquainted with the threats posed by computer malware of many different kinds. It's a brave personal computer owner who has no anti-virus protection, and more general anti-malware programs are becoming popular too. Many people, however, believe that this protection is both necessary *and* sufficient.

It's becoming increasingly apparent that Identity Theft is a burgeoning problem (see [2] for some data from early 2015), and in an age when wireless Internet connectivity is considered essential, as is being able to have it whilst out-and-about, it is here that we are the most vulnerable.

These vulnerabilities present challenges to the developers of mobile applications and connected devices, too, since they have a responsibility to protect their customers' information. In this article I want to consider some of the risks with this continuous connectivity, on the basis that fore-warned is fore-armed. To finish off, I'll suggest some simple things you can do to mitigate or eliminate those risks.

Threat landscape

The Age of Mobile Communications is in full swing, no doubt about it. Mobile phones have been with us for a good while, but much more recently, those phones have become 'smart', which essentially means 'connected to the Internet'. Although the take-up of technology such as 4G has been good (by the look of it), the coverage in the UK is still very patchy, and so many people take advantage of cafés, hotels, coffee shops and pubs providing customer Wi-Fi – often for free. Naturally enough, this has led to an increase in the use of these establishments as temporary offices, establishing a kind of barter trade of connectivity for beer/coffee/snacks which seems to be largely tolerated on both sides.

Many of us routinely allow our telephones, tablets, laptops, etc., to just connect to any wireless signal going, and use that channel to catch up on email, social media of various types, The News, whatever. With all of this wireless communications traffic saturating (in some cases, literally) the air-waves, it's no wonder, then, that folk with less than entirely honourable intentions have taken advantage.

Public Wi-Fi is only one part of the problem. The Internet of Things is a much less visible network of connected devices, and demonstrably vulnerable to attack (for some fun examples, see [3], and [4]). At least part of the reason that so many security flaws are found in such devices is that they are designed to be small, and have low power requirements, which translates to having insufficient processing power and/or resources to spare for things like encryption.

The home front

You might think that being able to crack your home router would be the crown jewels for most miscreants (and you shouldn't underestimate how bad that would be), but it's sufficiently difficult, if the security is good enough, that if an attacker can't get in with 'password', they'll *probably* give up. The norm these days is for home routers to use WPA2 with a pre-shared key – which is good enough as long as the password is sufficiently strong. (WEP security for routers is known to be deeply flawed. [5])

Cracking WPA2 protected Wi-Fi

It can be done with some inexpensive hardware, some open-source software, and a decent knowledge of networking at a low level. The hardware is a Wi-Fi receiver that can be put into what's known as 'monitor mode', which listens to all traffic on a channel even before an association is made with an access point. A packet sniffer can capture this traffic, including the handshake between a device and the access point. This handshake contains a hash of the passphrase used to protect the channel. Weak passphrases are susceptible to dictionary attacks – essentially a lookup table of pre-computed hashes to plaintext passwords. While this is extremely processor intensive, there are open-source tools available to automate it. In the end, good length (more than 12 characters), non-word passwords will probably defeat it.

The primary goal of an attacker is usually financial: if they cannot gain direct access to your bank account (and they almost certainly cannot do that), then the aim is to obtain enough information about you to be able to impersonate you well enough to transfer your money by convincing your bank that it is you authorising it, or spend your money by clocking up online transactions using your credit card or other means.

Of course, there are the obvious bits of information that are of direct interest – credit card numbers and security codes, your father's middle name, and so on – but *any* information is valuable, and can be obtained in a variety of ways. Your online presence in social networking, online petitions, mailing lists, just about anything, could provide someone with at least part of the means to pretend to be you, but hijacking your home router probably isn't the most effective way to obtain it. If I were an attacker, I wouldn't bother with targeting a single individual unless I knew it was going to be worth it. I'd much rather cast a wide net, and see what I could find.

By which I mean sitting in a public place when it's busy, and letting all that traffic come to me.

Hackers' delight

My mate Norm is a reasonably tech-savvy fellow. He lives in a fashionable part of town near to his work where he does 'something in IT', which is mostly keeping a small company email and web server in good order, and making sure the backups get run. It's not unusual on a Friday early evening to see him join some friends in one of the many bars that litter the route between the office and home, and tonight is the turn of a pub called The Queen's Head. He orders his round of drinks at the bar, and while he's

VERTICES
Vertices likes joining the dots between programs and devices, believes we're all being watched, and can be contacted at vertices@perfectcobalt.com

waiting, he takes out his phone and checks to see if there's a wireless network in range so he can try and round up a few more people to make a night of it.

Sure enough, he finds a nice strong signal from an access point (AP) broadcasting its name as 'QueensHead Customer Wifi'. It's an open network, so he doesn't have to go to the bother of asking for the Wi-Fi password, or seeing if it's written up anywhere, so he connects and is away on the Internet.

Is Norm taking unnecessary risks by doing this?

This being an open network, the traffic is unencrypted, and could – in theory – be 'sniffed' by anyone. This only matters, of course, for sensitive information being transmitted or received, e.g. login credentials, passwords, the stuff that Norm knows he wants to keep private. In practice, most or all of this traffic will be encrypted *before* it reaches the network, most commonly with a HTTPS connection from a browser or mobile-app versions of his social networking sites.

So, while an eavesdropper on the network can inspect the traffic, it'll still be encrypted. The security isn't perfect (there really is no such thing), but it's sufficiently difficult to crack the channel encryption that it will defeat anyone but the most determined and skilled attacker, who probably doesn't care about Norm's friends' updates on social networks, to be honest. (And if you think you **are** the target of such people, you already *know* how to protect yourself better than I do.)

Is it enough for Norm to depend on that level of security? What are the risks?

No, I'm Brian

Anyone can set up a wireless base station as an access point. Most modern mobile phones have the ability to do it, and even to 'tether' that AP to the mobile data connection out to the Internet. To continue the prior example, what do you think would be the effect of me doing that on my own phone in the same pub, and renaming my portable Wi-Fi hotspot to be the same as the open network – 'QueensHead Customer Wifi'?

Well, to be honest, in practice, not much effect at all. Firstly, most mobile phone hotspot facilities are deliberately WPA2 protected, so impersonating an open network wouldn't have much point, since anyone connecting would have to provide a password. Secondly, when devices such as *your* phone look for a network to connect to, they will inspect certain metrics about any in range. The most obvious of those metrics will be signal strength, and the pub's router providing 'real' Wi-Fi will be *much* stronger than my phone.

Still, it's a possibility, and in theory at least, if someone managed to connect to my wireless hotspot, I could intercept and inspect their traffic. And there is certainly no difficulty in having two devices broadcasting the same network SSID – there are perfectly reasonable reasons to do that. However, anyone who did connect would be using my bandwidth, and a mobile phone is *not* a great platform for packet sniffing.

A laptop of reasonable spec, however, is a different matter. It's fairly straightforward in Windows or Linux to advertise yourself as an access point with SSID, if your wireless hardware supports it. Inexpensive wireless cards and USB dongles are available which support the 'Master' mode required to do it, in any case, and having one of these also gets around the problem that you can't operate a wireless network device in Master mode *and* as a client to the Internet at the same time. Having two wireless interfaces means you can use one as an access point and the other to connect to the Internet. That's important, because not many people who connect to your access point will stay connected long if they can't themselves connect to the Internet!

What I've just described is what's known as a Rogue AP, and is a classic Man-in-the-Middle (MitM) attack using wireless networking. If you have connected your device to my AP, there are a few things I can now do to try and collect more interesting data than the possibly encrypted packets you're sending and receiving.

Before we go on, I must make clear at this point that doing *any* of these things is **illegal** in most countries. Specifically, intercepting somebody else's network connection without their permission, for any purpose, can land you in real trouble with the authorities. I describe them here for educational purposes. In short, don't do any of these things. YHBW.

Thought police

I can set up my own DNS server, and respond to any DNS requests by you with an address of my choosing – and therefore, with content of my choosing. That content could even be a look-alike page for the request you've made, requesting that you log in. I'm sure I don't need to spell out the consequences of that.

This attack is called a DNS Hijack, and it's made possible with a small handful of open-source tools and some networking know-how. [6] There's a good chance you've seen one in action, if you've ever used a public (open or not) Wi-Fi and been redirected to a 'Captive Portal' that requires a login, payment, or just acceptance of T&Cs. It's usually done the same way, by intercepting DNS requests and forwarding to the IP address of the portal. You might think this usage is benign, but of course it can itself be spoofed, and used to phish for credit card details, or other information.

Bait and switch

I have already mentioned that HTTPS is secure enough to deflect most attackers, but with a bit of ingenuity it's possible to subvert it under some circumstances. First, I can intercept your request for a secure (HTTPS) connection, and act as a proxy to your requested site, providing you with a regular HTTP connection, and maintaining the HTTPS connection myself, forwarding all requests and responses appropriately. As far as the remote site is concerned, you are connected securely, but you might notice that your own connection was HTTP only – if you happened to look, for example, at the browser address bar. This method might be used to obtain your credentials over an insecure line (between you and my router).

warn the user very loudly if an encrypted connection is unavailable

In a more sophisticated variant of this attack, my proxy could provide you with an SSL certificate, which would most likely cause a warning on your device that the connection might not be secure, but if you ignored that warning, you'd *think* you had a secure connection.

This attack is called SSL-stripping, and is described in detail on the website run by the man who developed it [7].

The coders' challenge

If you are writing code for mobile applications or IoT devices that process user information *in any way*, then you have a responsibility, and probably a legal requirement, to protect that information. As a developer, you need to be aware of actual and potential threats to make your product(s) secure. There is more to this than hardening code against buffer over-runs and SQL injection, although those things are still important.

Identifying those vulnerabilities, and crafting a viable attack to exploit them, actually requires a significant amount of effort, and won't even be considered if you're sending your admin password – or worse, your *customer's* password – in plaintext back to 'Mother'.

One of the simplest ways to be less insecure is to gather less data: if you don't need it, don't ask for it. However, I am aware that useful tools often require a revenue stream. The increasing appetite of marketing departments for our most personal data is unnerving at best (to me, anyway), but if you must have it, at least use it and transmit it securely. If your app talks over the Internet, make it always use HTTPS or other secure channel, and warn the user **very loudly** if an encrypted connection is unavailable.

Fortifying applications and devices with certificate authenticated security is not a trivial exercise, certainly it's harder than statically identifying buffer over-runs, but there it is. Fruit that hangs low on one side of the fence isn't necessarily easy to reach from the other.

Whoever you want me to be

You might at this point still regard these attacks as pretty unlikely. After all, they depend on you connecting to the Internet through an **open** access point that happens to be under my control, and conducting your business over that connection. I still have a couple of tricks in my bag of goodies to share. First, let's have a quick tour of how most devices associate with Wi-Fi routers.

Beacon and probe

When you turn on Wi-Fi on your phone (for example), you will often see a list of available wireless network names (SSIDs). This list is obtained by listening for Beacon Frames, which are part of the IEEE 802.11 management protocol. The beacon is sent by a wireless access point, and includes its name, MAC address and what (if any) security protocols it supports, amongst other things.

Conversely, your phone will send probe frames to listening APs. Broadcast probes are just a request for the list of networks in range, a direct counterpart of the beacon frame above. However, many devices *remember* the networks to which they have previously connected in a preferred network list. The device will probe directly for those networks (usually only when there is no active wireless connection). If the phone receives a response to such a probe, it will probably attempt to automatically connect (this is the default behaviour for most wireless device drivers).

If the phone is connected to a wireless network, and that connection is dropped, the preferred network list will be probed again. The response to a probe contains information about signal strength and data rates, and most devices will use this data to choose the 'best' connection.

I am Spartacus

I've already mentioned about wireless equipment and monitor mode, which allows me to listen to all and any wireless traffic including the management frames such as beacons and probes. If I receive a probe for a network called 'Spartacus', I can arrange to respond with the equivalent of 'I am Spartacus', and if I am broadcasting a strong signal, there is every chance the probing device will try to connect to my rogue access point.

One last thing to note: most devices will stop probing for new networks once they have associated and connected with an access point. However, with the right equipment – it's that inexpensive wireless hardware again – I can even arrange to impersonate the network to which you're currently connected, force you to drop that connection (this is called a de-auth, AKA 'get off my LAN'), at the same time as listening for the resulting probes and impersonating those networks too.

Moreover, my access point can impersonate *dozens* of networks simultaneously, capturing the traffic from anyone who allows their device to connect.

Are you paranoid yet?

Back to my question about Norm. *Now* do you think he was being reckless to depend just on HTTPS encryption over an open network? There are other attacks made possible (or easier) with the rogue access point, including session side-jacking, cross-site scripting and cross-site request forgery. If you think this is cause for concern, so you should – it definitely is! See [8], [9], [10], [11] and [12].

The real question now is this: is there anything you can do about it, to either detect the attacks, or render them useless to an attacker?

The answer to that is simple: yes there is. First, use a modern browser that implements HSTS [13]. This will defeat SSL stripping and protocol hijacking, and was in fact developed as a direct response to the original SSLStrip demo by Moxie Marlinspike (see [7]). Second, use a browser plugin such as HTTPSEverywhere [14] or one of its variants. This tool attempts to *enforce* a secure connection if one is available, and tries to make sure that your whole experience is as secure as it can be.

Lastly, invest in a Virtual Private Network, and use it anytime you're using a connection you cannot **guarantee** is safe. Doing all your Internet activity over the VPN (in *addition* to using HTTPS and everything else) renders most of the attacks I've described here toothless. DNS hijacking is easily defeated by knowing the IP address of your VPN server and not connecting by name, but a fake captive portal might be harder to bypass.

Using a VPN that you trust means that it *doesn't matter* if you're unwittingly connecting to the Internet via a rogue access point. The VPN server itself will usually provide its own end-points for DNS and gateways to the Internet, and most VPN software also encrypts the data transmitted and received, affording you an extra level of security.

Detecting the presence of a Rogue AP isn't necessarily easy, and in a corporate environment warrants expensive internal routing equipment and an army of vigilant network supervisors. However, on a personal level, if you notice that your phone is connected to a network with the same name as your home Wi-Fi, and you're in a cafe miles from your house, then unless your home Wi-Fi is named something common, it's probably a fake network. Similarly, if you find yourself connected to an open network that you were expecting to be protected, there's a good chance it's a rogue. Be wary of captive portals, and always be vigilant about personal information you provide to anyone.

Socially speaking

If I can get enough information on you, such as, say, your date of birth, post code and mother's maiden name, along with details of your bank account, I might be able to persuade your bankers to transfer money into an account I control. In fairness to most banks these days, they're more aware of the need for better security over the phone, and so this may not be enough – but if I am an attacker and can get even *some* of this data on you, it's a start.

If I can't use that information directly to get your money, I may be able to sell it on to someone else who *can*. Identity fraud is big business, attracting serious money, and correspondingly scary people. If I can (theoretically) do all this, then someone else with a real intent to rob you can *actually* do it.

If you ever use a Wi-Fi connection in a coffee shop, pub, hotel, airport, someone else's house, and especially if you ever use public open Wi-Fi, you need to be aware of the risks, and know how to protect yourself. Absolute security is a myth, of course, but you can do some things for little or no cost to make life much more difficult for any attacker. *Not* doing the simple things described here just make life unnecessarily easy for people who want to scam you or steal your identity.

If you are a company making mobile apps, or devices that use the Internet, whether it's a router, smart TV, home-automation IoT widget, or a toy, or you're a programmer writing the code for such things, if you make use of customer details then this stuff matters. If I am your customer, then it's not *your* data. It's *my* data. And it's just possible that in the future, I won't just have to take your word that you're secure, I'll be able to look for myself. [15] ■

References

[1] Fred Cohen, Wikipedia, https://en.wikipedia.org/wiki/Fred_Cohen

[2] Cifas, 'Identity Fraud up by 27%...', https://www.cifas.org.uk/id_fraud_first_quarter

[3] Nick Feamster, 'Who will secure the Internet of Things?', Jan 2016, https://freedom-to-tinker.com/blog/feamster/who-will-secure-the-internet-of-things/

[4] John Leyden, *The Register*, 'Hopelessly insecure Motorola CCTV...', Feb 2016, http://www.theregister.co.uk/2016/02/03/motorola_cctv_iot_insecure/

[5] Lee Barken, 'WEP Vulnerabilities', Dec 2003, http://www.informit.com/articles/article.aspx?p=102230&seqNum=12

[6] Chris Sanders, *Windows Security*, 'Understanding Man in the Middle Attacks', Apr 2010, http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Man-in-the-Middle-Attacks-ARP-Part2.html

Groovy and Grails eXchange 2015

Ralph McArdell reports on his conference experiences.

A week or two before Christmas 2015 I attended the 2015 Groovy and Grails eXchange conference thanks to the generosity of one of the speakers, ACCU member and occasional speaker at ACCU London meetings, Schalk Cronjé, who donated his complimentary conference ticket to me. The conference [1] was held over two days at the Skills Matter CodeNode building in central London on Monday and Tuesday the 14th and 15th of December 2015.

As I knew very little of Groovy and less of Grails my focus was on finding out a bit about Groovy, Grails and related topics. I did find out a quite bit but of course still have a long way to go so please accept my apologies for any errors and mistakes I may have made in what follows.

Monday

The conference registration opened a bit early at 08:00 so the keynote, titled 'Call me Apache Groovy' and given by Cédric Champeau, could commence at 09:00. The keynote was a mostly non-technical look at the history of Groovy and the community around it from its inception in 2003/2004 by James Strachan up to the 'present' of 2015 when Groovy became an Apache hosted project – hence 'Apache Groovy' – and one Russell Winder, a name some may recognize, is listed as a project committer. The talk ended with some speculation for Groovy in 2016 and beyond.

Following a quick break I went to a talk on SDKMAN [2] given by its creator Marco Vermeulen. SDKMAN was previously known as GVM (Groovy enVironment Manager) and manages Groovy installations, a task complicated by the combinations of available versions of Java related software development kits (SDKs). It seems SDKMAN can manage parallel installations of primarily Java related SDKs, has a command line interface and uses tools such as Bash, curl and unzip and PowerShell on Windows. SDKs are made available on the SDKMAN server by 'vendors' and once an SDK release is published on the server it is available to the SDKMAN client to download and install.

The second session of the morning I attended was given by Schalk who talked about idiomatic Gradle and recipes for Gradle plugin authors. I learned that there are over 600 Gradle plugins in the repository. Schalk noted that some are not that well written and proceeded to present a set of points to consider, and preferably follow, when writing plugins for Gradle.

I rounded off the morning with a talk given by Tony Davidson and Fergal Dearle on AST (Abstract Syntax Tree) transforms which used the building of a simple game engine DSL (Domain Specific Language) for kids as an example. They noted that even Hello World is not easy when trying to teach kids Java and how much easier children found Groovy – so much so they thought of the game engine DSL as a way to take the children further. I learned that Groovy supports AST transforms allowing modification of a Groovy compilation AST.

The first session I attended after lunch was on Groovy DevOps in the cloud given by Andrey Adamovich. Andrey noted that automating the provisioning of containers and servers is needed but doing so with Java is difficult. He then went on to describe various server provisioning automation scenarios involving, variously, ant, Gradle, Puppet and the like and ending up with a plethora of tools he has written using Groovy to automate such tasks including Gramazon [3] for controlling Amazon EC2 resources, sshoogr [4] for working with remote servers via SSH and p-unit [5] for verifying the results of provisioning scripts used with the likes of Puppet et al. He concluded that Groovy with Gradle makes the ultimate automation glue.

Thinking that it is always good to know how people go about testing the second talk of the afternoon I went to was given by Jeff Brown on testing in Grails 3. It turns out Grails 3 testing is performed using some Grails specific extensions to the Spock framework [6]. Being outside the community Spock was new to me but the examples that were presented looked to be quite elegant.

The final session of the day was a 'Park Bench Discussion' in which a bunch of the speakers sat up front and answered questions from the floor. Following the discussion there was a 'party' involving pizza and some free beer.

Tuesday

The second and final day of the conference started later than the first at 10:00 with a keynote given by Graeme Rocher on Grails 3.1 and the road ahead. Graeme started by running through the state of current Grails releases: that the 2.x series is in maintenance mode, version 3.0.10 was the then latest release and Grails plugins continue to be ported to the new 3.x architecture. Next came a recap of some salient points of Grails 3.0 particularly that a new architecture is used which is not a drop in replacement for Grails 2.x. Graeme then got to the meat of the matter and went over some of the goodies in Grails 3.1 which were mostly to do with

RALPH MCARDELL

Ralph McArdell has been programming for more than 30 years with around 20 spent as a freelance developer predominantly in C++. He does not ever want or expect to stop learning or improving his skills.

An Open Question (continued)

[7] Moxie Marlinspike, SSLStrip, http://www.thoughtcrime.org/software/sslstrip/

[8] Shaun Nichols, *The Register*, 'How to hijack MILLIONS of Samsung Mobes...', Jun 2015, http://www.theregister.co.uk/2015/06/17/samsung_flaw_keyboard/

[9] Chris Sanders, *Windows Security*, 'Understanding Man in the Middle Attacks', May 2010, http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Man-in-the-Middle-Attacks-ARP-Part3.html

[10] Scott Helme, 'Advanced Session Hijacking', Aug 2013, https://scotthelme.co.uk/advanced-session-hijacking/

[11] Wikipedia, https://en.wikipedia.org/wiki/Cross-site_scripting

[12] OWASP, Oct 2015, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[13] Wikipedia, HTTP Strict Transport Security, https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

[14] HTTPSEverywhere, Electronic Frontier Foundation, https://www.eff.org/https-everywhere

[15] Kieren McCarthy, *The Register*, 'Show us the code...', Jan 2016, http://www.theregister.co.uk/2016/01/25/source_code_ftc_commissioner/

Fifteen Love

Baron M sets a new puzzle for students of curious conundrums.

Greetings Sir R-----, come warm your bones at my table by the hearth and join me in a glass of brandy! I trust that winter's chill has not cooled your ardour for wager?

Good fellow! Splendid fellow!

I have in mind a game much played by the short-statured, excessively hairy and nature-obsessed folk of the Wimbledon commune, whose encampment I chanced upon whilst taking a morning constitutional some several years ago. I was drawn to it by a rather odd smell in the air, which I subsequently discovered to emanate from a peculiar blend of tobacco that they were especially fond of; a blend quite inferior to the Russian I can tell you!

When not dreamily pontificating on the virtues of love and natural living, they would gather up discarded periodicals with a view to beautifying their surroundings, and by a curious process make of them building materials for their accommodations, such as they were. Observing them at their labour, I suggested that rather than stoop to collect them they might make easier work of it by spearing some several of them at a time with a rapier.

That they eventually came to set upon folk peaceably going about their perambulations and confiscating their periodicals before they might have discarded them was a consequence that I contend no soul could possibly have foreseen!

But I digress!

Here, I have laid out the ace to nine of hearts upon the table (figure 1) which we shall take turns adding to our hands, with your picking first. With the ace counting as a one, if you can play a trick of three cards that add up to fifteen before either I can (figure 2) or we run out of cards then you shall have a coin from my purse. If not, then I shall have one from yours.

When I described this game to that weasel of a student with whom I am most regrettably acquainted, he made the truly bizarre claim that with the aid of wizardry the outcome of the game might be made certain. Now, as you well know, I have travelled widely and witnessed many truly astonishing events, but I cannot accept that that wretch has the slightest inkling of the magical arts! Although I suppose that one so bereft of wit might exaggerate his own skill, even to himself!

But let us talk of him no further. Here, take another glass and consider your strategy! ■

Figure 1

Figure 2

Courtesy of www.thusspakeak.com

BARON M

In the service of the Russian military the Baron has travelled widely in this world, and many others for that matter, defending the honour and the interests of the Empress of Russia. He is renowned for his bravery, his scrupulous honesty and his fondness for a wager.

Using Clara to Parse Command Lines in C++

Malcolm Noyes demonstrates how to get up and running.

Recently I needed a command line parser. Previously I have used Boost.ProgramOptions which has some nice features but this wasn't available in the build of Boost libraries I was using. I needed something simple and preferably header only. Some of you may know that I'm a bit of fan of Phil Nash's Catch as a C++ unit testing framework; Catch uses Clara internally to do the command line parsing, so I thought I'd give that a try...what follows is a brief description of the basic features, which hopefully will fill a few holes in the documentation!

Help!

Listing 1 shows a simple program that I'll use to demonstrate some features of Clara.

This example has three options. The first is to output help text and is introduced with `cli["-?"]["-h"]["--help"]`. This tells Clara that there are three ways to request help, two short options (`-?` and `-h`) and one long one (`--help`). If we run that we get something like this:

```
c:\Projects\ClaraExample\Debug>ClaraExample -h
usage:
  ClaraExample  [options]
where options are:
  -?, -h, --help describes how to use the program
  -s, --send     should the article be sent?
  -r, --reviewer <who to send to>
                 who should review it?
```

The help text gives us the options defined, along with the description that we supply to the `describe` clause. For an option with no additional arguments (`-s`) this is all we get, but for an option that requires input, we get the option together with the 'placeholder' text that we specified in the `bind` clause.

For the option with no arguments, I have 'bound' this option to the `ArticleOptions` bool member variable `send`. If I set this option, Clara will set the member variable...nice and simple.

The option that requires an argument I have bound to a string. If the argument is set then the member variable will take the value of the input. Let's see how that works:

```
c:\Projects\ClaraExample\Debug>ClaraExample -sr
"Phil Nash"
Send article for review by...Phil Nash
```

There are several ways to specify the input to an option that requires one; all these achieve the same result:

```
ClaraExample -s -r="Phil Nash"
ClaraExample -s -r:"Phil Nash"
ClaraExample -s --reviewer "Phil Nash"
```

Listing 1

```
#define CLARA_CONFIG_MAIN
#include <clara.h>
#include <iostream>

struct ArticleOptions {
  ArticleOptions() : showHelp(false),
     send(false) {}
  std::string processName;
  bool showHelp;
  bool send;
  std::string reviewer;
};
int main(int argc, char* argv[])
{
  using namespace Clara;
  CommandLine<ArticleOptions> cli;
  cli.bindProcessName
    (&ArticleOptions::processName);
  cli["-?"]["-h"]["--help"]
    .describe("describes how to use the program")
    .bind(&ArticleOptions::showHelp);
  cli["-s"]["--send"]
    .describe("should the article be sent?")
    .bind(&ArticleOptions::send);
  cli["-r"]["--reviewer"]
    .describe("who should review it?")
    .bind(&ArticleOptions::reviewer,
       "who to send to");
  ArticleOptions opt;
  cli.parseInto(argc,
     const_cast<const char**>(argv), opt);
  if (opt.showHelp) {
    cli.usage(std::cout, opt.processName);
  }
  else {
    if (opt.send) {
      std::cout
        << "Send article for review by..."
        << opt.reviewer
        << std::endl;
  }
  }
  return 0;
}
```

Did I ask for that?

Often I found that I wanted to know if a particular option was set and take some action based on that. Also, I wanted some default value to be set if the option wasn't set. It seems the best way to do that is to bind the option to a function (for example, see Listing 2).

This tells Clara to call the function if the option is set; the called function can do what it likes so in this case I can set both the flag and the value. If the flag is not set, then I use the default initialisation of the member variable. So now we have:

```
c:\Projects\ClaraExample\Debug>ClaraExample
Option not set...review by...Steve Love

c:\Projects\ClaraExample\Debug>ClaraExample
--reviewer "Phil Nash"
Send article for review by...Phil Nash
```

Positional arguments

If your program requires more than one argument, you can do that too. First, you can bind the option to a specific position, as in Listing 3.

MALCOLM NOYES

Malcolm Noyes has worked as a software developer/author for several years, and wrote several string classes before discovering the STL. He has never written a Unit Test framework but probably would have done if Phil Nash hadn't got there first.

Listing 2

```
struct ArticleOptions {
    ArticleOptions() : showHelp(false)
       , send(false)
       , reviewer("Steve Love")
    {}
    // ... as before ...
    bool send;
    std::string reviewer;

    void reviewerOptionSet(const std::string& v){
      send = true;
      reviewer = v;
    }
};

int main(int argc, char* argv[])
{
  //... as before ...
  cli["-r"]["--reviewer"]
    .describe("who should review it?")
    .bind(&ArticleOptions::reviewerOptionSet
       , "who to send to");

  //... as before ...
  if (opt.send) {
    std::cout << "Send article for review by..."
    << opt.reviewer << std::endl;
  }
  else {
    std::cout << "Option not set...review by..."
    << opt.reviewer << std::endl;
  }
}
```

Listing 3

```
//...
cli[1]
  .describe("other reviewer")
  .bind(&ArticleOptions::other,
    "other reviewer");
cli[2]
  .describe("another reviewer")
  .bind(&ArticleOptions::another,
    "another reviewer");

//...
  std::cout << "Option not set...review by..."
    << opt.reviewer << ", " << opt.other << ", "
    << opt.another << std::endl;
```

```
c:\Projects\ClaraExample\Debug>ClaraExample
"Phil Nash"
Option not set...review by...Steve Love,
Phil Nash,

c:\Projects\ClaraExample\Debug>ClaraExample
"Phil Nash" "Roger Orr"
Option not set...review by...Steve Love, Phil
Nash, Roger Orr
```

Note that these are not bound to a specific named option, so we still specify an option if we want to:

```
c:\Projects\ClaraExample\Debug>ClaraExample
"Phil Nash" -r "You know who" "Roger Orr"
Send article for review by...You know who,
Phil Nash, Roger Orr
```

On the other hand, if you don't care where an option should appear, the option can be bound to an 'unpositional' variable, as in Listing 4.

Listing 4

```
struct ArticleOptions {
  //...
  std::string omnificent;
};

  //...
  cli[_]
    .describe("if all else fails")
    .bind(&ArticleOptions::omnificent,
      "seek help from above");

  // ...
  std::cout << "If that fails, review by..."
            << opt.omnificent << std::endl;
```

```
c:\Projects\ClaraExample\Debug>ClaraExample
"Phil Nash" "Roger Orr" -r "You know who" God
If that fails, review by...God
```

Bah humbug...but I'm lazy...can't Clara figure that out?

One thing I would like to see is that Clara could do the test for whether an option was set; after all, Clara 'knows' whether the option was set since it either parse it or it didn't. That would save me having to have a flag for each option and possibly could be tested with something like:

```
if(cli['-r'].wasSet()) {
  ...
}
```

That would mean I wouldn't need to bind to a function that sets a flag and a value; instead I could just bind to the member. The lookup for the option would take longer than just testing the flag but I doubt if testing command line options is a time killer for most applications.

The other thing that I'd like is for the 'placeholder' text to be the default value, even if the option is not set. Then I wouldn't need to initialise the member in the struct, I can do it in the bind, which would improve the locality of the information associated with the option, something like Listing 5, which would be used like this:

```
c:\Projects\ClaraExample\Debug>ClaraExample
-v "Option overide"
Value set: Option override

c:\Projects\ClaraExample\Debug>ClaraExample
Not set: this is the default value
```

I'll have to ask Phil if he thinks either of those is a good idea...

Summary

Clara is very simple to get going; just download the header (1). It is header only, so there's no linking to incompatible libraries. What it does, it does very well and what it does less well can be worked around very easily. In short, it solved my problem, so I'm (mostly) happy! ■

Listing 5

```
struct Opt
{
  std::string value;
};
//...
cli["-v"]
  .bind(&Opt::value,
    "this is the default value");
//...
if(opt.wasSet("-v")) {
  std::cout << "Value set: " << opt.value
            << std::endl;
} else {
  std::cout << "Not set: " << opt.value
            << std::endl;
}
```

Software Development Is...

Pete Goodliffe defines the art, science, craft (and child's play) of software development.

And this, our life, exempt from public haunt, finds tongues in trees, books in the running brooks, sermons in stones, and good in everything.
William Shakespeare, As You Like It

It's a sad fact that I won't be able to rely on my sharply honed intellect forever. Some time in the future my wits will fade, and I'll no longer be the sharp, erudite, humble genius I am now. So I need a pension plan, a way to make my millions so that I can live in luxury in my old age.

My original plan for world domination seemed so simple it couldn't fail: *fizzy milk!* However, before I got a chance to work out the finer details of the recipe, I received devastating news: fizzy milk had already been invented. Gutted, and with the patent rights slipping through my fingers, I went back to the drawing board to come up with a new pension plan. And this time it was a good one.

This piece of genius goes back to the classic foods of my youth: custard and Alphabetti Spaghetti. I'm sure you can see where I'm going: *Alphabetti custard!* My initial experiments have proved promising. And almost palatable: it's a bit like rice pudding, but wheatier. Admittedly, it's an acquired taste, but I think it could catch on.

This software (food)stuff

Too much modern software is like my Alphabetti custard: it's *the wrong thing*, written *the wrong way*.

To make Alphabetti custard the 'right' way you'd make the pasta first by hand, and hand-mix a custard. The cheating, wrong way would be to buy tins of pasta, wash the sauce off, and then pour instant custard over the top.

One is a recipe, a method for reliable construction. The other is, at best, an adequate way to prototype, but not a large-scale fabrication technique.

As conscientious software developers, we should all aspire to write *the right thing* in the *right way*. One of the key characteristics of truly excellent programmers is actually caring about the software that we write, and how we write it. We need more lovingly baked artisanal code, no more of this tinned spaghetti nonsense.

So let's peer into the saucepan to investigate the nature of the software we write, and how we can avoid writing alphanumeric spaghetti ourselves. I'll pose a series of questions along the way to apply the lessons we learn. The first being: Do you want to improve as a programmer? Do you actually want to write the right thing in the right way?

If your answer is "No" then give up and stop reading now.

So, what *is* software development? To be sure, it's complex, with many interweaving aspects. It is part science, part art, part game, part sport, part chore, and more.

Software development is...an art

A great programmer needs to be, in part, a great artist. But is programming *really* an art? This is a debate that has long been held in software development circles. Some people think that programming is an engineering discipline, some an art form, some sit in-between, considering it a craft (I did call my first book *Code Craft*, after all).

Knuth is probably the most famous proponent of software as art, naming his famous series of books *The Art of Computer Programming*. He said this: Some programs are elegant, some are exquisite, some are sparkling. My claim is that is it possible to write grand programs, noble programs, truly magnificent ones! Stirring stuff.

There's more to code than bits and bytes, more than brackets and braces. There's structure and elegance. There's poise and balance. There is a sense of taste and aesthetics.

> A programmer needs good taste and a sense of aesthetics to write exceptional code.

There are many parts of the software development process akin to the creation of a work of art. The process is:

Creative

It requires imagination. The software must be skilfully constructed and precisely designed. Programmers must have a vision for the code they are about to create, and a plan of how they will make it. Sometimes that involves a great deal of ingenuity.

Aesthetic

Good code is hallmarked by elegance, beauty, and balance. It stands within the framework of certain cultural idioms. We consider the code's form alongside its function.

Mechanical

As any artist, we work in our particular medium with our particular tools, processes, and techniques. We work under commission for generous benefactors.

Team-based

Many forms of art are not single-person endeavours. Not every art form sees an artist sitting alone in their studio slaving day and night until their masterpiece is complete. Consider master sculptors with their apprentices. Consider the orchestra, each member held together by the conductor. Consider a musical composer, writing a piece which will then be interpreted by the performer(s). Or the architect designing a building that will be erected by a team of builders.

In many respects, the skill set of an artist is similar to that of a programmer.

Michelangelo was the archetypal renaissance man: a painter, sculptor, architect, poet, and engineer. Perhaps he would have made an incredible programmer. When asked about how he created one of his most famous works, the statue of David, he said: I looked into the stone and saw him there, and just chipped away everything else.

Is that what you do? Do you reduce and remove the complexities of the problem space, chipping them all away until you reach the beautiful code you were aiming for?

Here are a few questions to ask yourself on the theme of software as art:

- Do I consider the creative aspects of software development, or do I treat it as a mechanistic activity?

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

- Should I develop a keener sense of elegance and aesthetics in my code? Should I look beyond what's functional and solves the immediate problem?
- Do I think that my idea of 'beautiful' code is the One True Opinion? Should I consider artistry as a team pursuit?

Software development is...a science

We talk about *computer science*. So there must be something vaguely scientific going on somewhere, mustn't there? It's probably fair to say that in most development organisations there is much less science and far more plumbing happening.

The archetypal scientist is, of course, Albert Einstein. He was not only a genius, but also one of the most quotable people there has ever been (which helps authors considerably). He said this:

> Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius – and a lot of courage – to move in the opposite direction.

That is really profound; inappropriate complexity is a real killer in most software projects.

Einstein was also an aesthete. He appreciated elegance and beauty in his theories, and aimed to reduce things to a coherent whole. He said:

> I am enough of an artist to draw freely upon my imagination. Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.

See, I told you he was quotable.

So if software development is like a science, what does that mean? It is (or, rather, should be):

Rigorous

We look for bug-free code that works, all the time, every time. It must work with all sets of valid input, and respond appropriately to invalid input. Good software must be accurate, proven, measured, tested, and verified.

How do we achieve this? Good testing is key. We look for unit tests, integration tests, and system tests. Preferably automated to remove the risk of human error. We also look for experiential testing.

Systematic

Software development is not a hit-and-miss affair. You can't aim to create a well-structured large computer system by randomly accreting blobs of code until it appears to work. You need to plan, design, budget, and systematically construct.

It is an intellectual, logical, rational process; bringing order and understanding out of the chaos of the problem space and the design alternatives.

Insightful

Software development requires intellectual effort and astute analytical powers. This is especially apparent when tracking down tricky bugs. Like scientists, we form hypotheses, and apply something akin to scientific method (form a hypothesis, work out experiments, run experiments, and validate the theory).

> Good software development is not *cowboy coding*, throwing down the first code you can think of. It is a deliberate, considered, accurate endeavour.

Based on that, ask yourself:

- Is my software always totally correct and completely accurate? How do I prove this? How can I make this explicit, now and in the future?
- Do I strive to bring order out of chaos? Do I collapse complexity in my code until there are a few, small, unified parts?
- Do I approach problems methodically and thoughtfully, or do I rush headlong into them in an unstructured way?

Software development is...a sport

Most sports require great skill and effort: tenacity, training, discipline, teamwork, coaching, and self-consciousness. Likewise, software development involves:

Teamwork

It requires the concert of many people, with different skills, working in harmony.

Discipline

Each team member must be committed to the team, and willing to give their best. This requires dedication, hard work, and a lot of training.

You can't get good at soccer by sitting on a couch and watching soccer training videos. In fact, if you do it with a few beers and a tub of popcorn, you're likely to get worse at soccer! You have to actually do it, get out there on the pitch with people, practise your skills, and then you'll improve. You must train – have someone tell you how to improve.

The team must practise together, and work out how to function as a whole.

Rules

We're playing to (developing to) a set of rules, and a particular team culture. This is embodied in our development processes and procedures, as well as the rites and rituals of the software team and their tool workflows (consider how you collaborate around things like the source control system).

The teamwork analogy is clearest with a sport like soccer. You work in a group of closely functioning people, playing a game by a set of well-defined rules.

Have you seen a team of seven-year-olds playing soccer? There's one small guy left back standing in the goal mouth, and every other kid is running around the pitch maniacally chasing the ball. There's no passing. There's no communication. There's no awareness of the other team members. Just a pack of children converging on a small moving sphere.

Contrast that to a high-quality premier league team. They operate in a much more cohesive way. Everyone knows their responsibility, and the team works cohesively together. There is a shared vision that they work towards, and they form a high-functioning, well-coordinated whole:

- Do I have all of these skills? Do I work well in a team, or could I improve in some areas?
- Am I committed to my team, willing to work for the good of everyone?
- Am I still learning about software development? Do I learn from others, and am I perfecting my team skills?

Software development is...child's play

For me, this observation seems particularly appropriate; I'm really just a child at heart. Aren't we all?

It's fascinating to see how children grow and learn, how their world view changes and is shaped by each new experience. We can glean a lot from the way a child learns and reacts to the world.

Consider how this applies to our software development:

Learning

A child is aware that they are learning, that they don't know everything. This requires a simple characteristic: *humility*. Some of the programmers I have found hardest to work with think that they know it all. If there's something new they need to know, they read a book and then presume that they're an expert. A total humility bypass.

A child is constantly assimilating new knowledge. We must recognise that if we want to improve, we must learn. And we must be realistic about what we do, and do not, know.

Enjoy learning, savour finding out new things. Practise and improve your craft.

> Good programmers work with humility. They admit that they don't know it all.

Simplicity

Do you write the simplest code possible? Do you reduce everything to the least complex form to make it easier to understand and easier to code?

I love the way kids try to get to the bottom of things, to understand things from their own limited perspective. They're always asking *why*. Take, for example, a conversation I had with my daughter when she was six: Daddy, why is Millie my sister? Because you're in the same family as her, Alice. Why? Well, because you have the same mummy and daddy. Why? Because, well, you see, there are the birds and the bees... Oh go and get a book! ... *(thinking)* ... *Why?*...

We should be constantly asking *why* – questioning what we are doing and the reasons for it. Seeking a better understanding of the problem and the best solution. And we should strive for simplicity in our handiwork. That is not the most *simplistic* 'dumb' code possible, but the appropriately non-complex code.

Having fun

If all else fails, there's nothing wrong with this. All good developers enjoy a little playtime. My office currently houses a unicycle and a makeshift cricket pitch.

With that in mind, we can ask ourselves:

- Do I strive to write the simplest code possible? Or do I type what comes to mind, and not think about commonality, refactoring, or code design?
- Am I still learning? What can I learn about? What do I need to learn about?
- Am I a humble programmer?

Software development is...a chore

A lot of our software development work is not pleasant. It's not glamorous. It's not plain sailing. It's just donkey work that has to be done to get a project completed.

To be an effective programmer, you mustn't be afraid of the chores. Recognise that programming *is* hard work. Yes, it's great to do a cool design on the newest product version, but sometimes you need to do the tedious bug fixing and grubbing around the old awful messy code to get a product shipping and make some money.

From time to time we must become software janitors. This requires us to:

Clean up

We must spot problems and address them; work out where breakages are and what the appropriate fixes are. These fixes must be made in a timely and non-disruptive manner. A janitor does not leave the unpleasant tasks to someone else, but takes responsibility for them.

Work in the background

Janitors do not work in the limelight. They probably receive little recognition for their heroic efforts. This is very much a supporting, not a lead role.

Maintenance

A software janitor will remove dead code, fix broken code, refactor and rebuild inappropriate workmanship, and tidy and clean the code to ensure that it doesn't fall into disrepair.

Ask yourself:

- Am I happy to do code 'chores'? Or do I only want the glamorous work?
- Do I take responsibility for messy code and clean it up?

Metaphor overload

We often construct metaphors for the act of software development. Many of the insights we glean can be informative. However, no metaphor is perfect. Software development is its own special thing, and the act of creating it is not entirely like any other discipline. It's still a field we're exploring and refining. Beware of making wonky deductions from bad comparisons.

Good code and good coders are born from a desire to write the right thing in the right way, not from the software equivalent of Alphabetti custard. ■

Questions

1. Which of the metaphors outlined here do you relate most clearly with? Which most accurately reflects your work at the moment?
2. What other metaphors can you construct for the software pursuit? (Perhaps gardening or shepherding.) What new insights do these reveal?
3. How would *you* make Alphabetti custard?

Standards Report

Jonathan Wakely reports from the latest C and C++ meetings.

Shortly after I wrote my last report [1], the C and C++ committees both met, back to back, in Kona, Hawaii. This report covers both, but as I only attended the C++ meeting the C news is based on notes from my colleague Martin Sebor. Any misrepresentation of that meeting will be my mistake.

Hawaii seemed nice, although I barely left the hotel where the meeting happened and would have liked to spend some more time there. The views of the ocean were very pleasant, and I did go for a walk in the midday sun to find a post office one day, but I'm sure Hawaii has more to offer!

The full minutes of the C++ meeting are on the WG21 website in the document N4558 [2] but this report will cover what I thought were some of the highlights. Much to my surprise, we got a variant compromise! Before the meeting I thought the different expectations of a variant type (similar to Boost.Variant) would cause endless debate, but following a very productive evening session in Kona everyone agreed on a design that still allows variants to enter an invalid state, but that state is not toxic and doesn't lead to undefined behaviour if you so much as glance at it.

If you've been following C++ standards work recently you'll know that much of the committee's output is in the form of Technical Specifications (TS), where more experimental features are being specified, without the full formality (and finality) of an ISO standard. Following Kona the second version of the Library Fundamentals TS was finished and sent out for national body ballot. That includes everything in the first version plus two new wrapper types (`propagate_const` and `observer_ptr`), a functor for negating the result of other functions (`not_fn`), a simple API for the C++ random number facilities (`randint`), algorithms for erasing from containers (`erase` and `erase_if`), numerical functions for calculating GCD and LCM (`gcd` and `lcm`), a helper for writing values to a stream with a delimiter, solving the 'fencepost problem' (`ostream_joiner`) and metaprogramming traits for logical operations (conjunction, disjunction, negation). A second version of the Parallelism TS has been started, which adds Task Blocks to the first version. The Ranges and Networking proposals have progressed from proposals to working drafts, which basically means we think they're in pretty good shape and can now be polished and start the process of becoming a TS. The Evolution Working Group (EWG) decided to go ahead with a TS on Modules, one of the most eagerly anticipated features for C++. There are already some prototype implementations in the Microsoft and Clang compilers, but following different designs. A TS will mean we get a single direction that all compilers can implement, and users will be able to use modules in cross-platform code.

EWG also made progress on unifying the unified call syntax proposals, and approved proposals on inline variables (allowing variables to be defined in headers without getting multiple definition errors [3]); making exception specifications part of a function's type [4]; and preprocessor predicates for feature-testing (Clang's `__has_include` macro for testing whether a header file exists [5]). That means you can expect those features to make it into some future C++ standard. There are discussions happening about specifying the order of evaluation in some expressions but that work will resume in Jacksonville.

JONATHAN WAKELY

Jonathan's interest in C++ and free software began at university and led to working in the tools team at Red Hat, via the market research and financial sectors. He works on GCC's C++ Standard Library and participates in the C++ standards committee. He can be reached at accu@kayari.org

The Core WG spent a long time thinking about what an object is, when a region of memory becomes a live object, when an object's lifetime ends and the memory region returns to raw bytes. A new function called `std::launder` was proposed, which is a no-op but informs the compiler that whatever it knew about an object at an address should be forgotten about, which can be used after using placement new to create a new object where another one used to be. After the meeting it was pointed out that some industries might have trouble convincing regulators that 'launder' is an appropriate identifier to have in their code. (A similar objection was raised to the function `corrupted_by_exception` used to check whether a variant has entered its invalid state, which I think is a shame as I was hoping someone would form a C++-themed death metal band with that name).

The week after the C++ meeting the C committee, WG14, met at the same location. The C meeting was 4 days long and attended by 20 people, with a much narrower focus than the many streams and wide-ranging topics at the C++ meeting. The minutes are in N1978 on the WG14 website [6].

WG14 are working on a fifth part of TS 18661, which aligns C with the latest IEEE 754 / ISO 60559 standards, work that has been taking place in a sub-group for some years. The new part of the TS might include support for floating point exceptions, something that WG14 has resisted doing for some time due to it being very different to traditional C error handling.

Michael Wong of IBM did a presentation on transactional memory, which is already the subject of a C++ TS. That was received positively, so C might get TM support as well. The CPLEX group presented their proposal for parallel programming extensions to C [7] which was also positively received, but needs more work before it would be ready to become a TS.

There was also discussion of a new C standard. ISO rules require renewing a standard every 10 years, so there are four years left to decide what should be included. It looks like there will be a "C1y" standard in a year or two, which will be C11 plus technical corrigenda (fixes for bugs in the standard). That will be followed by a "C2x" standard with more major changes. Possible changes being considered include: transactional memory; removing or restricting some sources of undefined behaviour (including some preprocessor changes that have already been approved for the C++ preprocessor, which in practice is almost always a single tool that serves as both the C and C++ preprocessor); improving the <threads.h> specification; and some clean up regarding volatile. There is also a suggestion to drop Annex K, which defines the bounds-checking interfaces such as strcpy_s. These are not widely implemented, and it's not even agreed that they are actually any safer or less error-prone than the original functions.

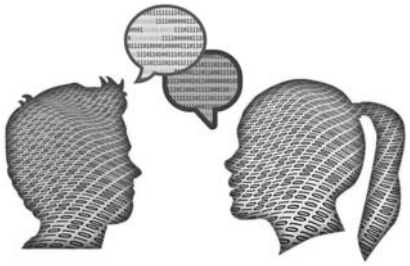
At the time of writing, the next WG21 meeting is the first week of March 2016 in Jacksonville, Florida, and the next WG14 meeting is mid-April in London.

References

[1] CVu 27-5, November 2015
[2] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4558.html
[3] http://open-std.org/JTC1/SC22/WG21/docs/papers/2015/n4424.pdf
[4] http://open-std.org/JTC1/SC22/WG21/docs/papers/2015/p0012r0.html
[5] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0061r0.html
[6] http://open-std.org/jtc1/sc22/wg14/www/docs/n1978.pdf
[7] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1966.pdf

Code Critique Competition 98

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



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. (We will remove email addresses!)

Last issue's code

I have a simple template class that holds a collection of values but sometimes code using the class crashes. I've written a simple test for the class, which works, but I do get a warning about a signed/unsigned mismatch on the **`for`** loop. I thought using **`auto`** would stop that. Is that anything to do with the crash? I've commented out all the other methods apart from **`add`** and **`remove`**.

The code is in Listing 1 (`values.h`) and Listing 2 (`test.cpp`).

Listing 1

```
#include <utility>
#include <vector>

#pragma once

// An unordered collection of values
template <typename T>
class Values
{
public:
  void add(T t);
  bool remove(T t);
  std::vector<T> const & values() const
  { return v; }
private:
  std::vector<T> v;
};

// Add a new item
template <typename T>
void Values<T>::add(T t)
{
   v.push_back(t);
}

// Remove an item
// Returns true if removed, false if not present
template <typename T>
bool Values<T>::remove(T t)
{
  bool found(false);
  for (auto i = 0; i != v.size(); ++i)
  {
    if (v[i] == t)
    {
      v[i] = std::move(v.back());
      v.pop_back();
      found = true;
    }
  }
  return found;
}
```

Listing 2

```
#include <iostream>
#include "values.h"

void test()
{
  Values<int> vi;
  for (int i = 0; i != 10; ++i)
  {
    vi.add(i);
  }

  if (!vi.remove(1))
  {
     std::cout << "Can't remove 1\n";
  }

  if (vi.remove(1))
  {
     std::cout << "Can remove 1 twice\n";
  }

  if (!vi.remove(9))
  {
    std::cout << "Can't remove 9\n";
  }

  std::cout << "size: " << vi.values().size()
    << std::endl;
}

int main()
{
  test();
}
```

Critiques

Mathias Gaunard <mathias@gaunard.com>

The answer is simple: **`pop_back()`** reduces the size by 1, so if **`i`** is already on the last element, when it enters the next iteration it is past the new size.

Two ways to solve this:

- return early once **`pop_back`** is done
- change **`!=`** to **`<`**

The test should also be amended to test removing the last element (it does remove 9, but by the time it does so it's not the last element anymore, since remove reorders).

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

Paul Floyd <paulf@free.fr>

To paraphrase Douglas Adams, we can get an answer to the problem, but we don't really know what the question is.

We have the comment telling us that Values contains an unordered collection, but are the items unique or not? If it is a multi-collection, should remove take out just the first instance or all instances? Furthermore, what are the complexity requirements for `values::remove()`?

I played around with the code a bit, and then read it a bit more carefully. There were three issues with `remove` that struck me:

1. If the item to be removed is the last in the vector, `pop_back()` will reduce the size by 1 whilst `i` will increment to the old `size()`, which is now the new `size()` plus one, missing the loop termination condition. The loop will continue reading from the end of the vector, doing bad things. Probably this is the cause of the crash described. This could be fixed by changing the `for` loop termination condition to be `i < v.size()`.
2. If the condition is true, then the item at `back()` jumps the queue to a position that won't be tested by the loop. So if the collection contains {0,1,2,1} and `remove(1)` is called, then when the first 1 is detected, the second 1 is moved to replace it and the next loop iteration continues with the 2. The second 1 thus evades detection. This could be fixed by decrementing `i` when an element is being removed so that the next iteration will test the moved value.
3. In the case where `t` (and thus `v[i]`) matches `v.back()`

   ```
   v[i] = std::move(v.back());
   ```

   is potentially performing self move assignment. This isn't a problem for the `int` type used in this example, but I believe that it is undefined behaviour for complex types.

   This could be fixed by adding a self assignment detection check.

Putting all that together, we have:

```
template <typename T>
bool Values<T>::remove(T t)
{
  bool found(false);

  for (auto i = 0; i < v.size(); ++i)
  {
    if (v[i] == t)
    {
      // avoid self move assignment
      if (i != v.size()-1)
      {
          v[i] = std::move(v.back());
          // ensure that the value at v.back()
          // that just got moved
          // is checked next time round
          --i;
      }
      v.pop_back();
      found = true;
    }
  }
  return found;
}
```

In practice, however, I would use the standard algorithms, for instance:

```
size_t oldSize(v.size());
v.erase(std::remove(v.begin(), v.end(), t),
  v.end());
return oldSize != v.size();
```

The complexity of the `erase`/`std::remove` solution may be quite different to the hand-rolled `remove` function. Since it preserves the order of the container, the worst case of removing 1 element from the front of the vector is that it has to move n-1 elements, which could be expensive. The hand rolled `remove` does not preserve order and may need to do fewer moves.

Robert Jones <robertgbjones@gmail.com>

My first thought on reading this Code Critique is that if at all possible I'd be inclined to try to avoid writing container classes. Users tend to have an expectation that any container will conform to the same style, and offer the same facilities, as the standard containers, which would make any container class a non-trivial exercise. If you're performing non-standard manipulations on a collection it may well be better to do it explicitly inline, so readers can see what's going on, rather than hide it behind the facade of a container class.

Coming to the code in detail, the first point is the signed/unsigned mismatch in

```
for (auto i = 0; i != v.size(); ++i)
```

which the author expected would have been eliminated by the use of `auto`. The mismatch occurs because the type of `i` is inferred from the initial assignment, which is with a literal '`0`', a signed value, which then doesn't match the type of `v.size()`, an unsigned value.

The smallest minimal change to correct this would be

```
for (auto i = 0u; i != v.size(); ++i)
```

In modern C++ this would really be better expressed directly using iterators, when the spurious signed type problem disappears

```
for ( auto i = v.begin(); i != v.end(); ++i )
```

and subsequently all occurrences of `v[i]` become `*i`

Rather more succinctly in C++11 we can just write

```
for ( auto & i : v )
```

and avoid the need to dereference at each usage. Notice that this syntax requires the introduction of the '`&`' l-value reference qualifier, otherwise '`i`' would always be a copy of the container value, not a reference to it.

Next we come to the question of the unexplained crashes.

All of the methods of iterating over the range implicitly or explicitly depend on the value of `end()` or `size()`, which is invalidated or changed from within the iteration loop by the use of `pop_back()` when the element is found, so the loop termination condition, based on equality, may never be met. One solution to this would be to break out of the loop when a matching element is found, however this would mean that only the first matching element is removed. Generally a better solution is the erase-remove idiom:

https://en.wikipedia.org/wiki/Erase%E2%80%93remove_idiom

Finally, the use of `std::move()` is creating more uncertainty. Thomas Becker covers this rather well in

http://thbecker.net/articles/rvalue_references/section_04.html

when he writes:

> Now consider the line
>
> ```
> a = std::move(b);
> ```
>
> If move semantics are implemented as a simple swap, then the effect of this is that the objects held by `a` and `b` are being exchanged between `a` and `b`. Nothing is being destructed yet. The object formerly held by `a` will of course be destructed eventually, namely, when `b` goes out of scope. Unless, of course, `b` becomes the target of a move, in which case the object formerly held by `a` gets passed on again. Therefore, as far as the implementer of the copy assignment operator is concerned, it is not known when the object formerly held by `a` will be destructed.

So the line in code `v[i] = std::move(v.back());` is potentially introducing an indeterminacy about when the value of `v[i]` will ultimately be destructed.

Generally the use of `std::move()` can be confined to constructors and assignment operators, so if you find yourself tempted to use it outside that context you want to think very carefully about the semantics of what you're writing.

James Holland <James.Holland@babcockinternational.com>

First, let's get the more obvious issues out of the way. The signed/unsigned mismatch warning can be resolved in several ways. As the student says,

`auto` has been used in an attempt to solve this problem. Unfortunately, it has not worked as the student had hoped. The problem is that `auto`, in this case, makes `i` the same type as `0` and `0`'s type is `int`. The type of such numeric literals can be changed to unsigned by adding the suffix `U` or `u`. Doing this will make `i` of type `unsigned int` (the same type of `size()`) and thus remove any mismatch. Another way to remove the type mismatch is to declare `i` of type `size_t`. `size_t` is unsigned, which is what is required, and makes the type of `i` clear.

Unfortunately the type mismatch has nothing to do with the crashing problem. It is also unfortunate that the student's test code does not highlight the problem with the code. What the test code fails to do is to attempt to remove a value from the (back) end of the collection. It is this that causes the crash as described by the following scenario.

Assume there are eight elements in the container and `remove()` is about to inspect the last element to discover whether it should be removed. In this situation the index `i` has the value 7 and `size()` returns 8. Further assume the last element is to be removed. The function `remove()` now moves the contents of the last element of the container to the element having index `i`. As `i` is an index to the last element, the last element is moved to itself. This is not necessarily a problem but what happens next is. The function `remove()` then removes the last element, making the size of the container 7, and increments `i`, making its value 8. When `remove()` determines whether there are any more elements to process, by means of the `for` loop, it discovers that `i` is not equal to `size()` (because 8 is not equal to 7) and so executes the body of the `for` loop again with disastrous consequences. Specifically, it attempts to access an element beyond the bounds of the container. The problem is that `i` has increased in value and the size of the container has decreased in value within one iteration. The `for` loop never saw their values being equal and so did not stop processing the container when required.

It may be tempting to change the `for` loop to execute when `i` is *less* than `v.size()`. This would stop the program crashing, but making this change would not correct another problem with `remove()`; it does not necessarily remove all the required elements. This can happen when more than one element is to be removed and one of them is at the back of the container. The function moves the element at the back of the container to one of the other elements being removed and then moves on, not realising that the moved-to element still needs to be removed. To cure this problem, the index `i` should not be incremented when a move is made. This will give `remove()` the opportunity to process the moved element appropriately.

A revised version of `remove()` that does not suffer from the crashing problem and removes all required elements is shown below.

```
template <typename T>
bool Values<T>::remove(T t)
{
  bool found(false);
  for (size_t i = 0; i != v.size();)
  {
    if (v[i] == t)
    {
      v[i] = std::move(v.back());
      v.pop_back();
      found = true;
    }
    else
    {
      ++i;
    }
  }
  return found;
}
```

As can be seen, the `for` loop is not now responsible for incrementing the index `i`. The index is now incremented only if the current element is not to be removed from the container.

The student has made `Values` a class template. This implies that the type of object that `Values` contains is not necessarily known. Such a type must be well behaved when copied to itself as this could occur during the execution of `remove()`. A type that uses heap based memory, for example, must not allow the heap to become corrupted. The easiest way to prevent this is for its copy assignment operator to check for self-assignment as shown below.

```
U & operator=(const U & rhs) noexcept
{
  if (this != &lhs)
  {
    ...
  }
  return *this;
}
```

A similar check should be made in its move assignment operator (`operator=(U && rhs)`).

One final point; `#pragma once` is not part of the C++ standard and so may not be supported on all platforms. The alternative is to use the standard include guard as follows.

```
#ifndef HEADER_H
#define HEADER_H
...
#endif
```

Commentary

There were a number of problems with the supplied code including the warning which was troubling the user. The warning was in this case a bit of a red herring as in this example the possible values over which `i` might range are small enough to fit exactly into the (signed) `int`. However, this is not always the case, and so avoiding the warning is good practice. As the critiques that tried to address this showed though, it is not as easy as you might like.

Changing `i` from `int` to `unsigned int` will typically double the maximum value `i` can reach – but the type of `v.size()` may well be larger than *either* of these (for example, it might be a 64-bit integer on a platform where `int` is only 32 bits) so removing the warning here actually hides a potential problem.

There was a long discussion on Herb Sutter's blog early last year – see http://herbsutter.com/2015/01/14/reader-qa-auto-and-for-loop-index-variables/ – which includes many different ways of trying to solve this particular problem!

The crashing, caused by iterating past the end of the collection if the *last* element is removed, is a specific instance of a general problem – it is easy to make mistakes when modifying a collection while iterating over it. In this example, if `remove()` needs only remove the *first* occurrence found, a safe solution is to `return` as soon as `pop_back()` has been called. In general though care must be taken or maybe the modification can deferred until after the iteration has completed.

There is some worrying code in the example when self-assignment occurs: `v[i] = std::move(v.back());` when both refer to the *same* element. While copy-assignment should be normally be designed to be safe under self-assignment – and to leave the item unchanged – there is no such requirement for self move assignment. The following simple program demonstrates the problem:

```
#include <iostream>
#include <vector>

int main()
{
  std::vector<int> v;
  v.push_back(1);
  v = std::move(v);
  std::cout << "size: " << v.size() << '\n';
}
```

Some compilers may print 0, others may print 1. It is also possible some might do other things... such as crash.

In this *particular* case it doesn't matter, as long as self move-assignment leaves the object in a valid state and does not invoke the dreaded 'undefined behaviour', as the very next thing the code does is to pop this item off the back of the vector and so destroy it. However, not all classes are necessarily going to produce defined behaviour so it is important, when using `std::move`, to ensure that the target is safe to treat as the *unique* reference to the object in this context.

Finally, as a couple of people pointed out, the program demonstrates the importance of checking that tests do actually test what is expected – the `remove(9)` *appears* to check removal of the last item but doesn't as by that point 9 is no longer the last item!

The winner of CC 97

There were four critiques which between them did a pretty good job of covering the various issues in the code.

I am aware that the problem was inadequately specified and Paul in particular explicitly raises this concern. In my professional experience, I find all too often that I am dealing with a bug (for example, as in this case, a crashing program) but the right solution is unclear as the specification of the function is incomplete. The missing ingredient with a code critique is being able to talk to the original author – but often that is a problem in a commercial environment too as the original author may have moved on.

Suggesting that the user look at standard solutions first is a good idea. Even if the complexity constraints are important, having a working simple solution can be a very useful first step in the right direction. I share Robert's concern that writing a container class can give people expectations about its behaviour that may not be the same as those of the author of the class.

Overall I think Paul asked good questions and covered the points well so he wins the prize for this issue's critique.

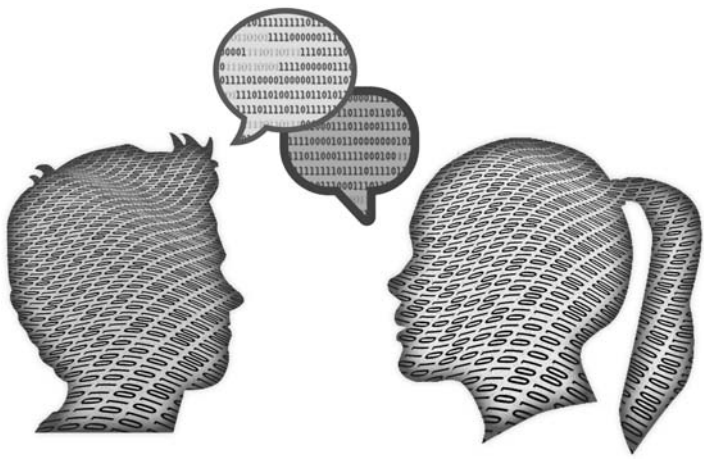
Code critique 98

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

> I am writing a simple program to index written text but it doesn't quite work. I want to print out every word in the document with a list of each line it appears on. I'm only getting the first occurrence listed but can't work out why.

Please explain why they have this problem... and suggest some other possible improvements to the program. The program is in Listing 3.

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



Listing 3

```
#include <iostream>
#include <map>
#include <sstream>
#include <vector>

int main()
{
  using namespace std;

  map<string, vector<int>> index;

  // read and index standard input
  string line;
  int lineno{};
  while (getline(cin, line))
  {
    ++lineno;
    istringstream iss(line);
    string word;
    while (iss >> word)
    {
      auto start =
        word.find_first_not_of(":;.,'\"?!-");
      auto end =
        word.find_last_not_of(":;.,'\"?!-");
      if (start != end)
        word.replace(end + 1, end, "");
        word.replace(0, start, "");

      if (word.empty()) continue;

      auto iter = index.find(word);
      if (iter == index.end())
      {
        index[word].push_back(lineno);
      }
      else
      {
        auto lines = iter->second;
        if (lines.back() == lineno)
          ; // ignore dups
        else
          lines.push_back(lineno);
      }
    }
  }
  // print the index
  for (auto entry : index)
  {
    cout << entry.first << ": ";
    string delim;
    for (auto line : entry.second)
    {
      cout << delim << line;
      delim = ", ";
    }
    cout << '\n';
  }
}
```

Bookcase

The latest roundup of book reviews.

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

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

Thanks to Pearson and Computer Bookshop for their continued support in providing us with books.
Astrid Byro (astrid.byro@gmail.com)

Make Games With Python

By Sean M. Tracey, published by MagPi Essentials, raspberrypi.org/magpi, Version 1

Free to download, hard-copy price to be announced

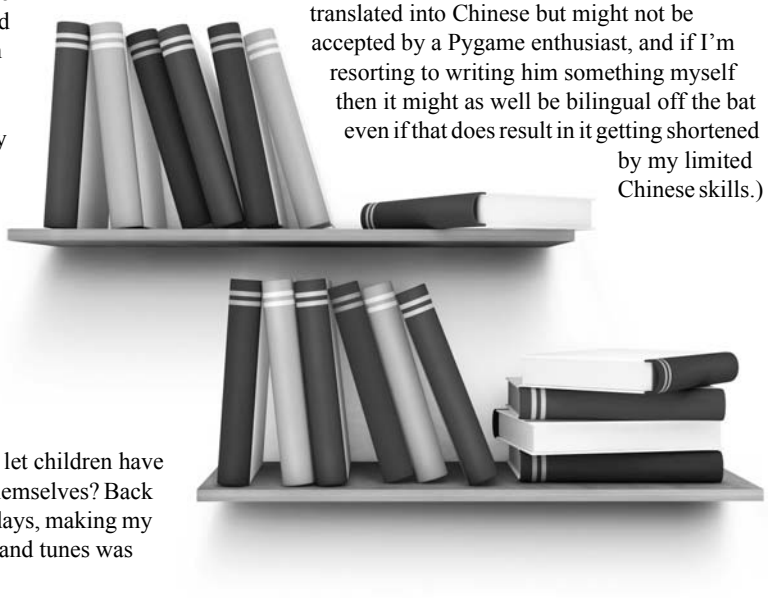
Reviewed by Silas S. Brown

I noticed this book on the Raspberry Pi blog at about the time I met a 9-year-old who is struggling with Pygame, and I was really hoping I'd be able to recommend it to him, especially as it's an official publication of the charitable Raspberry Pi Foundation, which I'm keen to help. Unfortunately, while I applaud the effort, the cause, and the price, I feel duty-bound to report I was not impressed enough to recommend giving this book to a child.

My worst gripe is how the book treats classes. After having made do with struct-like dictionaries for so long, classes are finally introduced in Chapter 8 with what appears to be a half-hearted attempt to explain why they're better (is a child who needs the word 'nemesis' to be defined really expected to understand 'it's not very semantic' without clarification?) and then the terminology gets more confusing: 'instantiation' is mentioned, but it's not made clear what an 'instance' is, and at one point we are told the player 'is a class' (not the sole instance of one) and, most confusingly, the player has an `__init__` method while the enemies do not: they have a factory function instead, and there's no explanation for this. But then we are told classes have 'functions': it doesn't even call them 'member functions', let alone 'methods', so how are we to tell which are inside the class and which are merely associated with it? Some of the design decisions are questionable too: in a shoot-'em-up, each enemy ship tracks its own bullets, then we are told that's inadequate because we want bullets to persist after ships are destroyed, and the 'fix' is to introduce a separate list of 'left-over bullets' and write code to transfer ownership of bullets upon a ship's destruction, but didn't anyone think of directly placing all bullets onto that list to begin with and doing away with the now-redundant per-ship lists? Or are we teaching children never to replace their earlier code?

In other news, a football held three feet off the ground gets only 1/9th the gravity it has at one foot. That's not what the physics chapter actually says, but its wording is vague enough to allow that interpretation. It could be fixed by explicitly stating that the distance which matters is that from the earth's centre, not its surface, but nobody seems to have thought of that. And then there's this clanger: `alien\'s are gonna kill me'. Yes, that apostrophe is wrong; no, there's no explanation of the escaping backslash or why we're putting this in a single-quoted string when all strings around it are double quoted, and no it doesn't help that quotes to the left of strings are printed as back-quotes without any note saying that's not what you're supposed to type (this is supposed to be a beginner's book).

On the subject of typing-in the listings, why is there so much dependence on sound and graphics files that you have to download separately? Not every child's Raspberry Pi will have Internet access. Showing how to load an image or play a sound from a file might be good, but, once that's done, I'd have transitioned swiftly to the kind of simple graphics and sound that children can type in themselves (and customise it as they go) rather than learning that they need to hire artists and professional musicians to do it – which may be true once you become a professional games coder, but why not let children have more fun doing it themselves? Back in the BBC Micro days, making my own sound effects and tunes was half the point!



Finally, there seem to be a fair few unexplained numbers in the middle of the code. I wouldn't go to the other extreme of creating a variable for everything, but in a book like this I'd expect a few more numbers to be given meaningful names, and perhaps even a little discussion (or at least a mention) of the design issue. The boy I met might just be trying to make a game, but I'm trying to sneak in a lesson or two about good programming practice while he's at it, and I'm not always sure the book's on my side.

The book's filename ends with _v1.pdf, implying this is version 1 and there will be future versions. Let's hope these address some of the issues. Meanwhile, for this particular boy, I wrote my own 'chapter' at http://people.ds.cam.ac.uk/ssb22/game.html for which I welcome criticism, although my decision to write all prose in Chinese as well as English did limit what I could say. (The book I mentioned on that page was not this one, but one he'd brought from China, and, although his parents are trying to make him go English-only, I felt OOP might be put at a severe disadvantage in that house if it doesn't get at least one explanation in Chinese to compete with the seductively-Chinese book that doesn't help. Francis Glassborow's *You Can Do It* has been translated into Chinese but might not be accepted by a Pygame enthusiast, and if I'm resorting to writing him something myself then it might as well be bilingual off the bat even if that does result in it getting shortened by my limited Chinese skills.)

View ~~from the~~ without a Chair

ACCU Committee
accu-committee@accu.org

Sadly, at the beginning of January Alan Lenton informed us that he was unable to continue as Chair of the ACCU and was standing down with immediate effect. The committee would like to thank Alan for stepping into the office of Chair in 2014 when the previous chair stood down: thanks Alan!

This unexpected development does however leave the association once again without a chair. Consequently this column represents some thoughts from a subset of the remainder of the committee. It also means that, with apologies, we do not have any details of issues for discussion at the AGM that Alan promised in what turned out to be his final View from the Chair column.

As well as Alan standing down our Secretary, Malcolm Noyes, has informed the committee that he will be standing down at the AGM in April. It is possibly not appreciated that any member can stand for any post and your association needs you! So please consider standing for the office of Chair or Secretary or any of the other offices or just as a committee member – and get in touch today [1]. By the time you get to read this all the proposals for the AGM will most likely have been published. Unfortunately this also means it will probably also be too late to accept nominations for officers or committee members before the AGM but due to recent changes in the rules you will be able to be proposed and nominated on the day of the AGM.

The AGM will as usual be held over lunchtime on the final day of the ACCU conference – although of course you do not need to attend the conference to attend the AGM. Should you have missed the various announcements about the conference, it is once again being held at the Bristol Marriot Hotel from Wednesday 20th to Saturday 23rd April with the usual pre-conference tutorial and workshop day on Tuesday 19th April with keynotes from Andrei Alexandrescu and Jim Coplien, as well as Anna-Jayne Metcalfe who you may recognise from her posts to accu-general and elsewhere. There are as usual too many interesting and intriguing looking sessions to be able to attend them all. Of particular note for this column is the session on Saturday morning devoted to ACCU issues – please come along and let us know your thoughts.

Links to details and registration can be found on the home page of the ACCU website [2].

As the saying goes it never rains but it pours and our long standing web master, Tim Pushman, has informed us that he is going to be retiring from web hosting and that we will need at the very least to move the ACCU website and its associated bits and pieces – such as the mailing lists – elsewhere.

While Tim, being a nice, professional and responsible person, is not pulling the plug immediately we do need to move relatively quickly on this and could really do with some help from any members who have web and hosting experience as those on the committee only have limited experience in these areas.

So if you have at least thought – if not expressed out loud – that the ACCU web site and services could be better here's your chance to make it so.

References

[1] ACCU Committee contact: accu-committee@accu.org
[2] ACCU website: http://accu.org

ACCU London: Dietmar Khül

A (clueless) review of the Jan 16th ACCU London Meeting.

Dear NHS Direct,

I recently attended ACCU London's talk about 'Quicker Sorting' by Dietmar Kühl, hoping to get my plumbing sorted or pick up some DIY tips. The evening started with a huge pile of burritos, which I was not expecting, though they were most welcome. The 30 or more of us then sat in the room and Dietmar proceeded to tell us how he had tried to beat `std::sort` starting with the undergraduate computer science exercise of writing quick sort. Having never been a computer science undergraduate, I was unclear how this exercise could possibly be good for anyone. I do prefer gardening, though my knees sometimes give me trouble.

Quick sort takes a pivot point, which is handy since I have recently had a pivot gate installed. I digress. Back to the plumbing. It then puts the smaller numbers on one side and the larger numbers on the other. And does the same for each side. This is apparently brilliant provided the numbers weren't already in order. I must wonder why you'd want to sort numbers that were already sorted, but people make all kinds of strange requests. Dietmar reassured us that it can sort all kinds of things whether they are numbers or not, and whether they are already sorted or not. I wonder if it can sort NaNs? I presume you could add a pre-condition check to see if the inputs were already sorted in order to avoid slowing things down by re-sorting them. But checking the precondition that might slow things down further.

So, quick sort isn't always that quick. Dietmar then showed how just comparing three or fewer items using `if`s and `else`s rather than recursing further speeds things up. He went on to show further refinements many of which got a few extra speedups, initially catching up with a standard implementation, and then beating it slightly.

The take home message was that `std::sort` is not a single algorithm and what they tell you in a computer science undergraduate degree might not be the whole story. He then took us to a German bar round the corner and we had a long discussion about climbing roses. Well I did, but I may have joined the wrong group in the pub.

Yours sincerely,

Mrs Trellis.

North Wales.

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?