the magazine of the accu

www.accu.org

{cvu}

Volume 26 • Issue 2 • May 2014 • £3

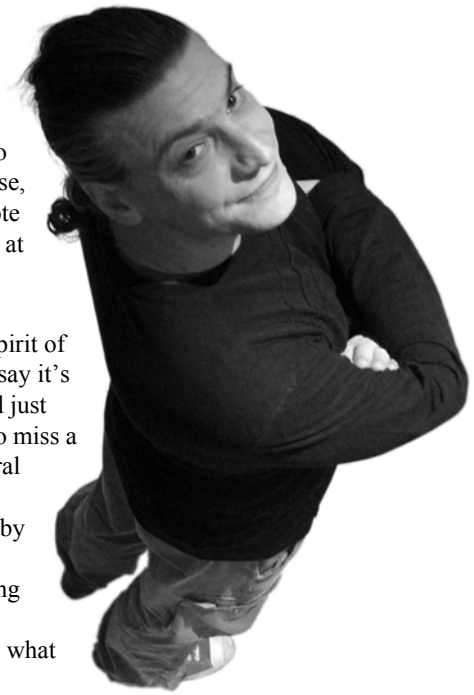
Features

Regulars

{cvu}
Volume 26 Issue 2
May 2014
ISSN 1354-3164
www.accu.org

accu

Continuous Learning



Another ACCU Conference passes, and another year when I wish I'd been able to see more of the sessions that ran parallel to those I attended. It's rarely an easy choice, deciding which to attend, but at least some of the sessions were video recorded, and slides are available for many of them. If I'm lucky, I'll get the chance to speak to people who went to different slots to the ones I chose, and then again, I'm fortunate that some people wrote up their own experiences of the conference so I can at least get a flavour of some of the things I missed.

Ultimately the indecisiveness associated with the conference is a positive thing! It reflects the very spirit of it – it is a programming conference. I'd hesitate to say it's *everything* about programming, because that would just take too long, and it's bad enough already having to miss a few sessions. It does cover a lot of ground, in several technologies, different programming languages, multiple disciplines and platforms, and is attended by a likewise eclectic bunch of people with genuine interest and enthusiasm on the subject. Programming isn't just about how best to represent some data structure in your language of choice, nor just about what tools and techniques are currently being used. Programming is an eco-structure that includes the coders, and architects, the managers, and the users.

What makes programming interesting for many of us is that it's an environment where it's impossible to know everything – and so is an opportunity to continually learn new things and discover fresh ideas. For me, that's why the ACCU Conference is so important: the diverse nature of the people attending, inspired by the variety of topics being presented and discussed, makes for a conference where it's easy to find out about new technologies, techniques and ideas, and have the opportunity to discuss and debate them with lots of well-informed practitioners.

STEVE LOVE
FEATURES EDITOR

The official magazine of ACCU

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

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

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

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

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

DIALOGUE

REGULARS

FEATURES

SUBMISSION DATES

C Vu 26.3: 1st June 2014
C Vu 26.4: 1st August 2014

Overload 122: 1st July 2014
Overload 123: 1st September 2014

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

An Ode to Code

Pete Goodliffe waxes poetic.

All bad poetry springs from genuine feeling.
~ Oscar Wilde

Gerald was a coder who worked in a small team.
The thing was: other coders coded code that was not clean.
The mess was detrimental, distracting, diabolic;
The inhumane detritus of an evil workaholic.

But Gerald had a conscience. He wouldn't let this lie.
He lay awake at night devising schemes to rectify
The awful internal structure, the confusing variable names,
And the contrived control flow that was consistently insane.

Those early days, 'The Boy Scout Rule' was how he planned to beat
The bugs and turgid software that had formed beneath his feet.
A tidy here, a bug fix there, refactors left and right.
Pretty soon, he thought, (with work) they'll all be out of sight.

But poor old Gerald, plan in action, missed one vital fact:
To make a dent, all programmers must enter in the pact.
His slapdash coding colleagues, just saw a rule to flout
And continued writing drivel whilst he tried to sort it out.

One step forwards, two steps back. Gerald danced this dance.
Until he learnt he needed a more militant stance.
Agile teams are excellent and clean code is the best.
To achieve this: the team, and not the code, would have to be addressed.

Conway's Law describes to us how software follows team –
Sympathetic software is born from a well-oiled machine.
If cogs get stuck or grate, and stop doing what they ought.
Then there's only one option: to remove them, Gerald thought.

So the team refactor started, with pattern: 'Parameterise from Above':
The manager, on his cycle home, received a surprise shove.
He landed down a manhole. You might call it homicide.
Gerald called it team hygiene. One problem had then died.

One by one his team mates met with unusual fates.
The unsuspecting QA team were hit by flying plates.
(The lesson learned from this event was: never hold team meetings
In a diner with bad furniture, and poltergeisty leanings.)

The programmers who caused such ire each met a gory end.
One 'caught his tie in the printer'; his face will never mend.
Another tripped atop the stairs on his way out for a break.
A pile of deadly Unix manuals flying in his wake.

Gerald's life was vastly better; the team was little more
Than one coder, a sys admin, and the guy who manned the door.
The problem with this setup, Gerald shortly found:
The code got no worse – good! – but it hardly changed,
as no coders were around.

Progress was slow and tough, though heroic Gerald tried.
Deadlines made a 'whooshing' sound as often they flew by.
With features sadly lacking, the project was a farce.
Then one day a policeman came, and put Gerald behind bars.

The moral of this simple tale is to react with care
When callous coder colleagues deign to drive you to despair.
The only sensible way there is to retaliate
Is British: maintain a healthy level of pent-up angst and hate.

Coding is a people problem

Hopefully you've read my article on ethics, so you probably agree that it is inadvisable to perform quite such a dramatic cull of the poorly performing members of your software team. However, how *should* you react when working with team members who do not perform adequately, or seem to wilfully make the code worse?

What if the software team leaders do not notice or comprehend the problem? What if, heaven forbid, they are part if the problem itself?

Sadly, at the bleeding edge of the codeface, this is not entirely unusual. Although some teams are full of awesome codesmiths, many are not. Unless you are unusually blessed in your coding career, you will at some point find yourself in sticky situations that seem to have no solution.

> Often, the tricky part of software development isn't in the technical aspects of the code; it's the people problems.

When the programmers just don't seem to get it, and fail to understand that they are making things worse, not better, you must respond.

Consider introducing practices that promote responsibility for the code and illustrate (in a way that avoids apportioning blame) how to work most effectively: introduce pair programming, mentoring, design review meetings, or the like.

Set an excellent example yourself. Do not fall into trap of those bad habits; it's very easy to lose enthusiasm and cut corners because everyone else is. If you can't beat them, *don't* join them.

> When surrounded by coders who do not care about the code, maintain healthy attitudes yourself. Beware of absorbing bad practices by osmosis.

It will not be simple or rapid to change a coding culture and steer development back towards healthy principles. But that doesn't mean that it can't be done. ■

Questions

1. How healthy is your current development team?
2. How can you quickly recognise when a developer is not performing as diligently as they should?
3. Which is most likely: people work sloppily on purpose, or they are sloppy because they don't appreciate how to work better?
4. How can you be sure that you're not adopting sloppy practices yourself? How can you prevent yourself from slipping into bad practices in the future?

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

Identity During Construction

Roger Orr observes some subtle traps in object initialization between different languages.

C++, Java and C# are related languages that share a lot of behaviour. However, one place where there are differences is in the finer details of how objects are constructed.

While this is not often an issue in practice, it can matter a lot and, when it matters, our often unconscious assumptions about 'how it works' can let us down.

To illustrate the main differences I'll work through basically the same example in C++, Java and C# so we can see how the behaviour changes.

C++

What would you expect to see from the code in Listing 1 that creates a single object of type `Derived`?

The way it works in C++ is that the object is constructed from the inside out.

First the `Base` sub-object is created; starting with the field initialisers and then calling the constructor. Note that while this constructor runs the object is of runtime type `Base` and consequently, the virtual call to `init()` calls the implementation in the `Base` class.

Then the `Derived` object is constructed, after starting with its field initialisers and then calling the constructor for `Derived` to complete the creation of the total object.

The full output is:

```
value()
Base ctor
Type=class Base
Base::i=1
Base::init
Base::i=1
value()
Derived ctor
Type=class Derived
Derived::j=2
```

Note too that the two calls to `value()` made to initialise `i` and `j` are each made just before the constructor body of the relevant class is executed.

The main advantage of this mechanism is that each 'layer' of the object can be created in turn and each one is completely constructed before the construction of the next layer starts. However, the consequential disadvantage is that a base class constructor (and correspondingly the destructor) has no knowledge of the existence (or type) of the complete object. This can be a nuisance when, for example, the values of fields in the base class depend on attributes of the complete object or when trying to display information about object life-cycle in a common base class.

A common workaround for the first issue is so-called 'two phase construction' when the object is constructed as usual and then a virtual method is invoked after the constructors have all finished to complete the initialisation of the object.

Listing 1

```
#include <iostream>
#include <typeinfo>
int value()
{
static int seed;
  std::cout << "value()\n";
  return ++seed;
}

class Base
{
private:
  int i = value();
public:
  Base()
  {
    std::cout << "Base ctor\n";
    std::cout << "Type=" << typeid(*this).name()
              << "\n";
    std::cout << "Base::i=" << i << "\n";
    init();
  }
  virtual void init()
  {
    std::cout << "Base::init\n";
    std::cout << "Base::i=" << i << "\n";
  }
};

class Derived: public Base
{
private:
  int j = value();
public:
  Derived()
  {
    std::cout << "Derived ctor\n";
    std::cout << "Type=" << typeid(*this).name()
              << "\n";
    std::cout << "Derived::j=" << j << "\n";
  }
  void init()
  {
    std::cout << "Derived::init\n";
    std::cout << "Base::j=" << j << "\n";
  }
};
int main()
{
  Derived d;
}
```

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

Java

Re-writing this example in Java produces the two classes in Listing 2 and the derived class in Listing 3.

The rules in Java are slightly different – in particular there is not the same rigid inside-out construction order and the object is of a consistent type

Listing 2

```
public class Base
{
  private static int seed;
  public static int value()
  {
    System.out.println("value()");
    return ++seed;
  }
  private int i = value();
  public Base()
  {
    System.out.println("Base ctor");
    System.out.println(String.format("Type=%s",
       getClass().getName()));
    System.out.println(String.format("Base.i=%s",
       i));
    init();
  }
  public void init()
  {
    System.out.println("Base.init");
    System.out.println(String.format("Base.i=%s",
       i));
  }
}
```

Listing 3

```
public class Derived extends Base
{
  private int j = Base.value();
  public Derived()
  {
    System.out.println("Derived ctor");
    System.out.println(String.format("Type=%s",
       getClass().getName()));
    System.out.println(String.format
       ("Derived.j=%s", j));
  }
  public void init()
  {
    System.out.println("Derived.init");
    System.out.println
       (String.format("Derived.j=%s", j));
  }
  public static void main(String[] args)
  {
    Derived d = new Derived();
  }
}
```

throughout construction even though, when in the base constructor, the output object is incomplete.

The complete output from this example is

```
value()
Base ctor
Type=Derived
Base.i=1
Derived.init
Derived.j=0
value()
Derived ctor
Type=Derived
Derived.j=2
```

The main difference from the C++ example is that when `init` is called in the `Base` class constructor the call goes to the method in the `Derived` class – even though the fields of this class are not yet populated (as shown by the line containing `Derived.j=0`).

Listing 4

```
public class Base
{
  private static int seed;
  public static int value()
  {
    System.Console.WriteLine("value()");
    return ++seed;
  }
  private int i = value();
  public Base()
  {
    System.Console.WriteLine("Base ctor");
    init();
    System.Console.WriteLine("Type={0}",
       GetType().ToString());
    System.Console.WriteLine("Base.i={0}", i);
  }
  public virtual void init()
  {
    System.Console.WriteLine("Base.init");
    System.Console.WriteLine("Base.i={0}", i);
  }
}
```

Listing 5

```
public class Derived : Base
{
  private int j = Base.value();
  public Derived()
  {
    System.Console.WriteLine("Derived ctor");
    System.Console.WriteLine("Type={0}",
       GetType().ToString());
    System.Console.WriteLine("Derived.j={0}", j);
  }
  public override void init()
  {
    System.Console.WriteLine("Derived.init");
    System.Console.WriteLine("Derived.j={0}", j);
  }
  public static void Main()
  {
    Derived d = new Derived();
  }
}
```

Additionally, if we change the code in the `init` method to modify the value of `j` the value written will be overwritten by the subsequent initialisation!

However, while perhaps not ideal, the behaviour is well defined since in Java (as in C#) all the fields in the object are zero-initialised at the start and hence the value of any field will be precisely specified.

If C++ allowed similar behaviour it would be very hard to avoid undefined behaviour as the fields in the derived object would be uninitialised (and typically unspecified) before their initialisation occurred.

C#

The initialisation of objects in C# is, unsurprisingly, a lot like that in Java but there are still a couple of interesting differences that can trap the unwary.

Listing 4 is a C# equivalent to the earlier examples (I've not followed the C# capitalisation conventions to try and make the code easier to compare with the examples from the other two languages) and the derived class is in Listing 5.

The initialisation in C# follows Java in that the object type remains the same throughout the calls to both the `Base` and the `Derived` constructor (and hence, as in the Java case, the call to `init()` in the `Base` constructor is made to the implementation in the `Derived` class.)

ACCU – For the Unknown Unknowns

Chris Oldwood ponders what the ACCU means to him.

In the previous issue of *C Vu* the Editor asked the question, "What is ACCU for?" I think each of us has different ideas about what they get, and, more importantly, want from being a member of ACCU. As Alan Griffiths has said on a few occasions in his 'From the Chair' in *C Vu*: the ACCU is what we, the members, make of it. Whilst I can't say for sure what I'd want to change, if anything, what I can say is what it means to me and how I use it. Your mileage will almost certainly vary…

I've written in these pages before about my own background and how I came to join ACCU [1]. In short, what I was looking for at the time was a replacement for the *C/C++ Users Journal* and *C++ Report* as they had both gone the way of the dodo. What I found took me by surprise because not only was there deep knowledge of C++, there was also a lot of discussion about the higher-order aspects of programming in general.

Whilst some might consider a detailed discussion [2] of a three-line C++ function that parses the surname from a comma-separated full name to be excessive, I personally found it to be one of the most thoughtful threads on the accu-general mailing list for some time. What it brought home was how much variation there can be in such a simple piece of code. Not only was the method of implementation open to question, with the obvious C++ 98/C++ 11 dimension, the choice of argument parsing convention and even the function name itself went under the microscope in an attempt to establish maximum clarity. Although this was more information than the original poster was after, for (some of) us bystanders it was an interesting code review.

In my experience you just don't get that attention to detail in the usual forums and newsgroups. Admittedly you don't want every discussion to descend into that level of deconstruction or we'd struggle to even write our supposed industry-standard 10 lines of code per day. But it's a useful exercise to go through every now and again to remind ourselves of all the various forces that we need to balance in our quest to produce simple, maintainable code.

Whilst I've reached a level of programming ability that puts 'the basics' into the area of Conscious Competence [3] I'm acutely aware that the range of my expertise defined by Conscious Incompetence grows rapidly bigger. At least the good news is that some of that is no longer filed under the more disturbing Unconscious Incompetence (or as it's more colloquially known: The Unknown Unknowns [4]). And that I largely put down to my membership of ACCU.

A common reply to the question of how to learn more about programming is that there are loads of blogs and forums out there – so just google it. It's a fair point, but it also misses the simple fact that to google something you have to be aware of its existence in the first place. Sometimes, as was the case with 'persistent data structures' I thought I knew what they were until Kevlin Henney's Immutability talk at the ACCU conference last week. There I discovered those people using them as a synonym for 'immutable container' were being economical with the truth. Although it's quite possible they don't realise it themselves.

Take programming idioms for example. One of the best methods to learn about new ways of working is to learn other programming languages, because they force you to solve similar problems in different ways. When you're the one attempting to establish a new idiom based on some other language it's easy, but when you're the maintenance programmer staring at some 'weird code' wondering what on Earth the author was thinking it's a different prospect. Sometimes there can be a fine line between genius and incompetence.

Have you ever had a problem, spent hours googling it, found the answer and then tried to google for the answer again based on what you now know and found the answer at the top of the results list? I find one of the biggest hurdles to finding the answer to some of the less technology specific programming questions is knowing the correct terminology to use in the question. What most of my initial googling tends to be is fumbling around trying to find the correct words to use for the right search query; once I have them the answer generally comes fairly quickly, or there are no results and its back to square one.

For me ACCU plays a large role in the serendipitous side of programming. My awareness of my programming surroundings comes largely from being a member of this specific organisation. There is also a time and a place for pedantry and I personally believe that more often than not its use within ACCU discussions has a positive rather than a negative effect. I want to understand and use the terminology of our profession (and the ideas embodied within them) with precision and, so far, I think being an ACCU member has been the most valuable way of achieving that. ■

References

[1] The Downs and Ups of Being an ACCU Member, *C Vu* 25, March 2013

[2] accu-general thread 'Clearest code', January 2014

[3] http://en.wikipedia.org/wiki/Four_stages_of_competence

[4] http://en.wikipedia.org/wiki/There_are_known_knowns

CHRIS OLDWOOD

Chris is a freelance developer who started out as a bedroom coder in the 80s writing assembler on 8-bit micros; these days it's C++ and C#. He also commentates on the Godmanchester duck race. Contact him at gort@cix.co.uk or@chrisoldwood

Identity During Construction (continued)

Where it differs is that the field initialisation occurs up-front before either constructor runs; and additionally the construction of `j` in the `Derived` class comes before the construction of i in the `Base` class – hence in this example the values of `i` and `j` are reversed from the values in both the previous languages.

Conclusion

The initialisation of objects (and their fields) is subtly different between C++, Java and C#. It is easy to make a mistake if you don't realise what the differences are, especially if you are converting code written in one language to run in another one.

The problem is that often, while the resultant code will compile and execute without reporting any problems, the behaviour may not be what is expected.

I hope this simple example of the basic behaviour may help people avoid making this category of mistake. ■

List Incomprehension

Chris Oldwood gets some data structure anxiety off his chest.

I've never really got on very well with the `list` data type. Back when I started programming professionally, I was writing code in C and we had to create many of the basic data types ourselves. Generally speaking, at the time, if you needed a variable sized collection you opted for a linked-list, and usually a doubly-linked list at that. This gave you more flexibility when it came to traversing, and also the same underlying structure could be used for a Stack or Queue depending on whether you needed to add or remove from the head or tail.

As a consequence I grew up for many years believing a list always had a head, a body and a tail; where the head and tail were a single item at the front and back respectively and the body was all the elements in between. The side-effect of this distorted mental model was a great difficulty [1] understanding the concept of `Typelist`s when Andrei Alexandrescu published his seminal work *Modern C++ Design*.

I lay some of the blame for my early programming ills firmly at the door of MFC. The following snippet shows what MFC's idea of a list class does (of course it also reinforces my own already misguided belief of what a list was at that point too):

```
CList list;

list.AddHead(42);
list.AddTail(99);

ASSERT(list.GetHead() == 42);
ASSERT(list.GetTail() == 99);
```

Around the same time as I was struggling to grasp what I later discovered was a 'recursive model' of lists (single item head + multi-itemed tail) I was also trying to learn Python. Unlike C++, where the default container type was an array (`std::vector`), Python was presented around the list type and terms like 'list comprehension' and 'cons operator' did very little to help me understand what was going on. It didn't help either that the list type smelt an awful lot like an array to me and further talk of slices and shallow copies just seemed a far cry from the safe world of C++ that I was used to.

It's probably no surprise I didn't really grok the list stuff in Python when I didn't exactly understand the true power of iterators in C++ either. For that light bulb moment to happen I had to read Matthew Wilson's *Extended STL* book. Up until that point iterators seemed to me mostly just 'clever' pointers, but what Matthew's book did was to show me the more the powerful idea of representing other types of 'collections' as sequences such as a file-system directory or a database table. For example, to iterate over the files in a folder using C++ 98 might go something like this:

```
FolderIterator it("C:\\Temp");
FolderIterator end;

for (; it != end; ++it)
{
  const std::string& filename = *it;
  ...
}
```

At around the same time I joined ACCU and the first local branch meeting I attended was in Cambridge. This was hosted by Jez Higgins and the title of the talk was 'Iteration: It's just one damn thing after another'. While I can't claim to have understood all the non-C++ variations it did begin to help cement my realisation that containers are often just used as temporary storage for consuming a sequence.

With this revolutionary (to me) mindset under my belt the eventual transition from C++ to C# was somewhat easier because its ubiquitous `IEnumerable` type felt very similar to an iterator in C++ and the C# `List<T>` type was really just a `std::vector<T>`. In fact it wasn't until I hit a performance anomaly in PowerShell that I even knew C# had a classic linked-list type.

Even with help from Boost's `iterator_facade`, implementing simple iterators in C++ are still a chore. In contrast the 'yield return' construct in C# makes it a doddle because the compiler builds a state machine under the covers for you so that your function can become the internal algorithm for an iterator:

```
public IEnumerable<int> NumbersTo10()
{
  for (int i = 1; i != 11; ++i)
    yield return i;
}
. . .
IEnumerable<int> range = NumbersTo10();
foreach (var number in range)
  sum += number;
```

Manually looping over a range like the numbers 1 to 10 is an algorithm that is fairly well ingrained in my muscle memory. When I first saw this way of doing it I was bewildered:

```
var range = Enumerable.Range(1, 10);
```

But of course it's highly unlikely you'd pass that to a `foreach` loop, you'd use LINQ all the way and hide the loop entirely:

```
var sum = Enumerable.Range(1, 10)
                    .Sum();
```

I still felt deeply uncomfortable with this. The classic `for` loop is simple and my C++ background meant I was fairly certain the (JIT) compiler would translate it into some equally simple machine code. With the Enumerable method I'm not so sure, it feels like there are a few memory allocations going on there which just seems excessive. No matter how much I might remind myself of the Parento Principle [2] (a.k.a. the 80/20 rule) it just seems gratuitous.

Despite attending Oliver Sturm's ACCU conference session on F# way back in 2009 and listening to Don Syme himself talk about F# to the BCS a couple of months later, it has taken me until now to try and pick it up seriously. I purchased a second hand copy of *Programming F#* and started reading it in the background mostly as a way of gently introducing myself to some of the functional programming concepts that I was aware of, like currying and partial function application, but didn't really know in depth.

And then on page 33 I hit that term 'list comprehension' again, but this time the example (slightly modified below) didn't seem all that confusing:

```
let numbersTo10 =
[
  for i in 1..10 do
    yield i
 ]
```

CHRIS OLDWOOD

Chris is a freelance developer who started out as a bedroom coder in the 80s writing assembler on 8-bit micros; these days it's C++ and C#. He also commentates on the Godmanchester duck race. Contact him at gort@cix.co.uk or@chrisoldwood

Debuggers are for Wimps

Frances Buontempo gives a quick lesson in debugging for Python.

Having spotted several team members tending to use a debugger to see if their code was working or not, I adopted the phrase "Debuggers are for wimps" and tried to encourage people to write unit tests to check small chunks of their programs were behaving. What I had failed to notice was that not everyone was using a debugger for python. This came to light when I went to work with a colleague who was taking a while troubleshooting a production issue. It became apparent he was copying and running the script one line at a time in the iPython repl [1]. This was very time consuming and probably served me right for denouncing debuggers. My amended phrase now reads "Debuggers are for wimps, unless you are trying to find a bug".

I wrote up this short lesson in the basics of using the python debuggers, pdb, which comes for free with python. Of course, some people will use IDEs which come with debuggers, and there are other debuggers. The python website gives a very long list of helpful tools [2]; however, it is worth spending a little time getting the hang of pdb which is suitable for simple debugging tasks. It is based a command line tool, based on the GNU debugger, gdb.

pdb – python debugger

If your script throws an exception it is very easy to run it under the debugger and see what caused the problem.

For example consider the following (rubbish) python program `myscript.py`:

```
def naughty():
  raise Exception()
naughty()
```

If we just run it the exception is unhandled and we see the following message

```
C:\Dev\src>python myscript.py
Traceback (most recent call last):
  File "myscript.py", line 4, in <module>
  naughty()
  File "myscript.py", line 2, in naughty
  raise Exception()
Exception
```

So, we know what line the problem's on. We could change the script to see what's going on by importing the pdb module and setting a breakpoint on the offending line.

```
import pdb; pdb.set_trace()
```

When we run it again the breakpoint is triggered when it gets to the line containing **`set_trace`**.

Of course, you may not want to change the code itself, no matter how small the change is. In that case, simply run the offending script through the pdb module

```
python -m pdb myscript.py
```

The **`-m pdb`** invokes the debugging module and sends it `myscript.py`. This will halt on an exception, allowing you to inspect the local variables and walk down the stack trace

Run script through debugger

Invoke the pdb module in python and send it your script, e.g.

```
python -m pdb myscript.py
```

This gives a **`(Pdb)`** prompt to type instructions in to, starting at the top of the script:

```
> c:\dev\src\myscript.py(1)<module>()
-> def naughty():
(Pdb)
```

Type **c** for continue – it will halt when it gets an exception, as seen in Listing 1.

At this point various commands can be issued. For example,

```
(Pdb) p <variable_name>
```

can be used to inspect variables, and

```
(Pdb) a
```

shows any arguments to a function. If you're lucky, this might be enough to spot your problem.

FRANCES BUONTEMPO

Frances has a PhD technically in Chemical Engineering, but mainly programming and learning about AI and data mining. She has been programming professionally for over 12 years. She can be contacted at frances.buontempo@gmail.com.

List Incomprehension (continued)

In fact it looked very similar to the C# equivalent. Suddenly the mystique was gone – a 'list comprehension' is just a list generated by some algorithm. I remember reading a tweet once suggesting 'list comprehension' was probably up there in the top 10 badly named computer science terms. To me 'sequence generation' seems a more suitable term.

But hold on, that term doesn't seem quite right either; perhaps 'list generation' would be more appropriate. But why would you generate a list, which takes up valuable space, from a lazily evaluate-able sequence which takes virtually no space at all, except as an optimisation for further iteration?

It feels like I've discovered the beauty of lazily evaluated sequences with iterators in C++ and **`IEnumerable`** in C# and now consequently discovered what list comprehensions are. But in return I'm now questioning why so much importance seems to be placed on them when sequences and pipelines appear to offer so much more value. Will my journey never end? ■

References

[1] http://chrisoldwood.blogspot.co.uk/2013/06/a-tail-of-two-lists.html
[2] http://en.wikipedia.org/wiki/Pareto_principle

ACCU Conference 2014

Chris Oldwood and Ian Bruntlett review the ACCU 2014 Conference.

Chris Oldwood gives his persepective

Another 12 months has flown by and I find myself heading down to Bristol for the ACCU annual conference. This will be my 7th year in attendance and 4th time speaking – I think I'm beginning to get the hang of this now.

Last year saw the move to the new venue in Bristol; this year it's more likely to be business as usual as we'll all know where the various rooms, bar and toilets are. Any teething troubles (not that I personally noticed any) will likely be ironed out and hopefully we can now just relax and enjoy the show.

Here's how it went for me…

Tuesday

Despite not going to any of the tutorials I was still treated to an unexpected talk by Uncle Bob as the hotel was also playing host to the Bath & Bristol Scrum Group. So, whilst not technically an ACCU event it was nonetheless a good start to my conference program. He gave a short session about the past and possible future of Agile, with a particular emphasis on Kent Beck's original notion of healing the divide between developers and The Business. This was a great reminder of why we do the practices we do.

Wednesday

The opening keynote came from Bill Liao who helped a school kid set up a not-for-profit coding dojo in Ireland to help teach kids programming. Naturally it's grown and has gone multi-national but the key point seems to be that it's run by, and for, kids which is a great achievement. The boy/girl ratio is almost equal in some cases and their mantra of 'Be Cool' has helped maintain a positive attitude. It was certainly an inspirational start to the week's proceedings.

Clearly Didier Verna's persistence has paid off because I chose Uncle Bob for my first session which was about how Clojure is the new C. The first half of his talk was pretty much a cut-down version of his Requiem for C keynote from a few years' back where he laments how far most of us are now abstracted from the hardware. Given this, his argument was that LISP, perhaps in the form of Clojure, provides a modern 'high-level assembly language'. While I agreed with many of his points I think I'll still find writing expressions in infix notion more convenient.

The lunchtime break was followed by Charles Bailey's talk on Git archaeology. Now that I've used Git for 9 months I felt prepared to tackle some of the more hardcore Git tools. After a brief recap of the Git object model he provided a number of examples to show how you can slice-and-dice the log to unearth useful details about the changes in your commit history. Making sense of merge commits is definitely essential and he

CHRIS OLDWOOD

Chris is a freelance developer who started out as a bedroom coder in the 80s writing assembler on 8-bit micros; these days it's C++ and C#. He also commentates on the Godmanchester duck race. Contact him at gort@cix.co.uk or@chrisoldwood

Debuggers Are For Wimps (continued)

Main Pdb commands

The manual [3] provides a complete list of all the commands. Aside from printing variables and arguments to a function, other important commands include

- **q** for quitting the debug session,
- **s** to step (maybe into another function)
- **n** to move on to the next line.
- **w** shows where you are in the stack,
- **u** moves up the stack
- **d** moves down.

You may want to set a breakpoint somewhere in your script. Simply load it via the pdb module, and before continuing with **c**, issue breakpoint commands: **b(reak) [[filename:]lineno**

Obviously, avoiding writing bugs in the first place would save even more time; however, spending a little time learning how to use the debugger can save hours. As stated, there are other debuggers for python, but pdb is relatively simple to use and can help you pinpoint a problem very quickly. ■

Listing 1

```
(  Pdb) c
Traceback (most recent call last):
  File "C:\Python27\lib\pdb.py", line 1314, in
main
  pdb._runscript(mainpyfile)
  File "C:\Python27\lib\pdb.py", line 1233, in
  _runscript
  self.run(statement)
  File "C:\Python27\lib\bdb.py", line 387, in run
  exec cmd in globals, locals
  File "<string>", line 1, in <module>
  File "myscript.py", line 1, in <module>
  def naughty():
  File "myscript.py", line 2, in naughty
  raise Exception()
Exception
Uncaught exception. Entering post mortem debugging
Running 'cont' or 'step' will restart the program
> c:\dev\src\myscript.py(2)naughty()
-> raise Exception()
(Pdb)
```

References

[1] http://ipython.org/
[2] https://wiki.python.org/moin/PythonDebuggingTools
[3] http://docs.python.org/2/library/pdb.html

provided some good advice there too after showing us some real-life octopus merges.

My final session for the day was spent with Roger Orr who was attempting show what Order Notation is, and more importantly, how it can be used in practice. With the mathematical definitions out of the way Roger, in his own inimitable way, took the seemingly obvious case of `strlen()` and showed how in extreme cases it can stop being `O(N)` and be much worse (think page faults). He then showed a more traditional use when tackling the many different ways you can create a sorted container of integers by using both sorted and non-sorted containers.

The 15-minute lightning talks were back again this year to provide a backdrop to the shorter 5-minute versions that run on Thursday and Friday. For the third time this week I got to see Uncle Bob, this time giving his views on what our Professional Disciplines should be if regulation were ever to rear its ugly head. Next up Nico Josuttis gave an impassioned plea to consider what is happening to our privacy and what we, as programmers, can do to help provide the tools to enable more secure communication.

I followed Nico with a (possibly deluded) attempt to cast code snippets in a slightly humorous modern-art style. Steve Freeman then continued the more classical lightning talk tone by explaining why the Given/When/Then style of tests can be unnecessarily verbose as we try to tackle the contorted language that often comes from writing this style of test. Finally Pete Goodliffe concluded the day with a short and amusing poem about code.

Thursday

Four years ago Dan North failed to make his conference keynote due to volcano problems. Luckily this year it went without a hitch as he tried to explain why Agile doesn't scale. This was an interesting and entertaining talk as he explained how the same principles we apply at team-scale just don't work when applied at the organisation level. Whilst I know first-hand of the effects of 'Enterprise Scale' development it was good to understand more about what (supposedly) drives the middle and higher tiers of management.

I chose Wojciech Seliga's talk to take me up to lunch which was about the problems faced dealing with the high volume of automated tests that covers a variety of different build configurations and platforms for their JIRA product. This was a really good talk as it covered both the quality of the tests themselves, the attitudes of the developers working on them and the performance of the build pipeline. There was lots of good stuff to take away and think about.

After a spot of lunch I decided to go with two shorter talks. The first was 'Continuous Delivery with Legacy Code' by Dan Swain & Leon Hewitt which was about the architectural refactoring work they had needed to do to support use of a secure 3rd party payment service. Whilst the narrative was very interesting due to the constraints imposed by the secure service, I felt the use of 'Continuous Delivery' in the title was somewhat misleading.

This talk was immediately followed by Sven Rosvall who showed how modern mocking tools can be used directly on concrete types and static methods as an alternative to introducing interfaces or proxies into your design. The tools do some low-level jiggery-pokery to the underlying virtual machine to work around the normal mocking limitations and he showed a couple of examples in both .Net and Java. Whilst it's good to have tools like this I wasn't the only one left wondering whether they would be used as an excuse to avoid refactoring or coming up with a better design.

My final session that day was to be with James Grenning. Even though I haven't written any C in 15-odd years I have worked on some OO-style C codebases and I was intrigued to see how he was going to apply the classic SOLID principles to a non-natively OO language. It all worked surprisingly well given the lack of built-in polymorphism and the Single Responsibility Principle always pays dividends. It's amazing how much unnecessary gunk we can encapsulate and avoid leaking in our header files. We then watched him write some C code TDD-style that made me wish I'd known that 20 years ago!

The first set of twelve 5-minute lightning talks got underway after a break with Francis Glassborow describing how important it is that programmers learn multiple languages of both the computer and natural kinds. The others included Charles Bailey pointed out Git's use of the colour red instead of green to denote deleted code, Thaddaeus Frogley did a primer on the graphics pipeline, Dirk Haun with a neat wearable camera, Andy Balaam wowed us with programs that generate themselves and Dmytro Mindra convinced us it's good for geeks to get up and talk.

Friday

Didier Verna provided us with the Friday keynote titled 'Biological Realms in Computer Science'. This was an engaging talk that delved into the similarities between Biology and Computer Science. The analogies between cells and computers were fascinating and he had many excellent quotes to draw upon to supplement his apparent fondness for the T-800 Terminator.

Whilst the vast majority of talks are obviously relevant to our day-to-day jobs as programmers there is still room for some more left-field sessions that address our other needs as a human being. Phil Nash's talk looked into the idea of healthy body/healthy mind to remind us that our effectiveness as a programmer depends heavily on the way we look after ourselves, or not. Instead of just looking superficially at our diet he delved right down into the chemistry to tell us in more detail about what it is we're eating. The background for this was his own impressive personal journey to look after himself better and in turn improve his cognitive abilities.

Two years ago at the ACCU conference I purchased a copy of *Overload* Issue 8 (amongst others) and it contained an article by one Kevlin Henney about immutability. That article provided the starting point for an excellent talk about how we can use immutability more effectively, especially when concurrency is involved. Jon Skeet predicated the Java Date class would feature at some point and he wasn't disappointed. He even did a good job of trying to convince us that editing his slides to fix a coding error wasn't a state change.

The late afternoon slot was the easiest choice to make of the whole event as it was time for my own talk. I knew I'd picked a contentious topic – version control practices – but luckily the flame-war was reserved for the bar afterwards.

Another short break and the second set of lightning talks were underway. Due to the hotel getting the main room ready for the dinner, we all squeezed into one of the smaller rooms and sadly many couldn't fit in! Once again there were twelve talks with Ed Sykes kicking off the set by explaining about the Quantified Dev project he's working on. Some of the other talks included Emyr Williams answering Pete Goodliffe's call to become a better programmer, Alisdair Meredith showing us what's coming up in C++ 14, Frank Birbacher describing colour spaces, a poem from Charles Tolman and a somewhat harrowing account of what happened in Ukraine from Dmytro Mindra.

At the beginning I said the big change happened last year, but there was another significant change this year – the conference dinner. For a start the menu was less traditional; which was not surprising given the Bollywood Banquet theme. The tables surrounded a dance floor and we were treated to some wonderful Bollywood music and dancing between courses. We also got to join in and strut our stuff, plus there was a 'dance off' between the US and UK, although I'm not sure exactly what prize victory brought.

Of course some things never change and we soon retired to the bar until far too late to chew over the week's events and life in general.

Saturday

I'm sure one day I'll grow up and go to bed at a responsible time during the conference. Until that day comes I'm probably going to keep missing the opening sessions on the Saturday morning (hint, hint, start them later) and with me leaving early I only got to see one talk. It was also the only C++ talk I attended – Anthony Williams on the future of C++ concurrency. This was a detailed look into some of the proposals that might feature in the next standard (at the moment some are already documented in a TS).

Anthony took us through continuations, executors and schedulers to show how we might compose, control and manage our task workload. Naturally `std:async` and `std::future` got to feature heavily, not least because of the contentious nature of their lifetimes.

On reflection

I've been out of the professional C++ programming world for a few years so that makes it marginally easier to choose my sessions because I don't feel duty bound to go to them all. Anything that can help narrow the selection process is a must if a decision is ever to be made at all. Dithering too long can be the difference between fitting in the room and having to start the selection process all over again!

As I sat on the train on the way back home I remembered how annoyed I was at having to put Breaking Bad season 4 to one side whilst I came to the conference. Clearly the four days were pretty intense because I never once thought about what else I might be doing instead (spending time with the wife and kids notwithstanding). It just goes to show that seven years on it still hasn't lost its magic.

Ian Bruntlett shares his experiences

This was my first Bristol ACCU conference. It's a long way from Morpeth to Bristol so I was concerned about the journey. It was also my first ACCU conference talk. I essentially came to the conference to 1) gain experience in public speaking and 2) to go to the conference talk. Due to my condition, my attendance at the conference had to be carefully planned.

Throughout the conference I picked up some goodies. Some memory sticks. Some t-shirts. Bought an ACCU 2014 conference t-shirt and a mug. Undo software had mugs with a programme with a bug on it in 64 bit Intel/AMD assembly language. I didn't find the bug in it but they gave me a mug anyway for being so persistent. Thank you Undo :)

Bloomberg were the main sponsors of the event and some of its employees did interesting talks. There was a Bloomberg suite with arcade games and a pool table, (sometimes) a free bar and, interestingly enough, PCs running Bloomberg's suite of software.

Some sessions in the conference were filmed with a view to putting them online at infoq.com.

I volunteered for a couple of things 1) to review a book *Expert PHP and MySQL* when I bumped into Asti who runs the ACCU book reviews and 2) offered to help out Boost as a volunteer documenter.

Monday 7th April

Travelled to Bristol by train. Went smoothly. Made my way to the hotel via taxi. No problems.

Tuesday 8th April – pre-conference tutorials

Nico Josuttis did a marathon (10:00 – 17:00) session, 'Switching to C++11 and C++14 in one day'. I got my main copy of his STL book signed.

Conferring/socialising

It was good to meet people who I only knew from e-mail or I had met already from previous ACCU conferences.

Wednesday 9th April – Day 1

Interesting talks 'Range and Elevation – C++ In a Modern World', 'There ain't no such thing as a Universal Reference', 'The C++14 Standard Library', 'The biggest mistakes in the C++ library', Lightning talks – too many to mention, all very interesting. All followed by the 'Welcome' reception.

Thursday 10th April – Day 2

Interesting keynote about Agile and its limitations. Interesting talks 'C++ Dynamic Performance', 'Large Scale C++ Performance' (first time I have seen John Lakos speak, very interesting), 'The art of Learning and Mentoring', Lightening talks – again too many to mention, all very interesting. Followed by a presentation by Bath Ales – interesting but I'm not a beer drinker.

Friday 11th April – Day 3

Keynote speech about 'Biological realms in Computer Science'. Interesting sessions on 'A healthy mind in a healthy body'. Kevlin Henney did a good talk, 'Immutability FTW!' which reminded me of some of his other talks about objects existing on three scales – Identity, State and Behaviour and Value Based programming. There was a talk about 'C++ Undefined Behaviour'. At the end of the night there was the Conference Dinner with a Bollywood theme. There were 3 female dancers and 1 male dancer and a dancing workshop as well. Not everyone was keen on this.

Saturday 12th April – Day 4

No keynote. Interesting talks – 'Executors for C++', 'Crafting more effective technical presentations'. Then my talk 'HOWTO – The Brain' (PDF available from the Conference website) followed by Astrid's 'Cultural Considerations for Working in the Middle East'. To finish the conference we had 'Everything You Ever Wanted To Know About Move Semantics (and then some)' – which went into detail about special member functions

Sunday 13th April

Travelled back to Morpeth. Nightmare. Trains were delayed, two of which were cancelled outright. Fortunately a friend gave me a lift from the station and I could finally relax. ■

Note: All the website slides/documents can be found at http://accu.org/index.php/conferences/accu_conference_2014/accu2014_schedule

IAN BRUNTLETT

On and off, Ian has been programming for some years. He is a volunteer system administrator for a mental health charity called Contact (www.contactmorpeth.org.uk). As part of his work, Ian has compiled a free Software Toolkit (http://contactmorpeth/wikispaces.com/SoftwareToolkit).

Standards Report

Mark Radford reports the latest developments in C++ Standardization.

Hello and welcome to my latest standards report.

You may recall that in my previous report I was covering the (ISO) C++ Standards Committee meeting that was taking place at the time in Issaquah, WA, USA. The post-Issaquah mailing is now available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/#mailing2014-03. Unfortunately I could only cover the first half of the meeting because, owing to the *CVu* publication timetable, I had to ship my report before the meeting concluded. Therefore, in this report, I will go back to that meeting and deal some with some of what happened in the second half of the week. Note that most of the papers referred to come from the pre-Issaquah mailing: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/#mailing2014-01.

SG1 (Concurrency and Parallelism) remained active for the rest of the week. In one of their sessions (on Wednesday morning) they discussed Arch Robinson's paper 'A Primer on Scheduling Fork-Join Parallelism with Work Stealing' (N3872). This paper is a bit different in that it does not propose anything. Instead, it explains two important design choices for code where a single thread splits (or forks) into two threads, and then the second thread rejoins with execution continuing on a single thread. This approach is useful when execution reaches a point where, in a sequence of operations, some operations can proceed in parallel with others. In this situation, one thread is said to steal work from the other thread. One important point that came out of the discussion was that if blocking (including blocking due to locks) is involved, it can affect which stealing decision is made (see the paper for details of the available stealing choices). There was agreement that more work is needed to include the effect of blocking in the paper.

SG1 are preparing a TS detailing a number of concurrency extensions for C++. On Wednesday afternoon they discussed 'Improvements to `std::future<T>` and Related APIs' (N3857). This paper describes a number of proposed extensions to `std::future` for inclusion in the TS. One of the potential improvements described in this paper is the addition of a `then()` member function, which allows another task to be scheduled following the execution of the one submitted to the `std::async` call. The ensuing discussion somewhat surprised me, because I thought the addition of `then()` was a done deal. However, reading the minutes from Wednesday afternoon in SG1, it looks like it may not be straightforward. There is a paper in the mailings entitled 'Resumable Functions' (N3858), and resumable functions achieve the same thing as `then()` via a more general mechanism. Some of those in the discussion made the point that resumable functions would make `then()` obsolete. A straw poll was taken on whether `then()` should go into the TS, the result being ten in favour and five against, giving a 'borderline consensus'. There was some more discussion but, as far as I can see, no firm conclusion was reached. Looking in the WG21 minutes (N3901) I see there is a LWG motion as follows: "Move to direct the project editor for the Concurrency Technical Specification to produce an initial working paper based on N3785 and N3857 and incorporating modifications suggested by the reviews at this meeting". Given the 'borderline consensus' I mention above, I'm not sure if `then()` will be included or not, but I will endeavour to find out in time for my next report.

Before moving away from SG1 I thought I'd just mention their Thursday afternoon discussion of Christopher Kohlhoff's paper 'Library Foundations For Asynchronous Operations' (N3896). This paper agues that futures can be a poor choice as a building block for asynchronous operations in C++. It goes on to present the case for a pure callback approach, on the grounds that it "allows efficient composition of asynchronous operations". Looking at the minutes, I think maybe there were some misunderstandings about the contents of the paper, probably because it is in need of more work. Some were concerned about the complexity involved, while others pointed out that the complexity would be in the library and would not leak into the users' code. This is born out by the conclusion to the discussion: SG1 agreed that the author should continue with the approach but should try to simplify the interfaces, or demonstrate why they can not be simplified. Also simpler examples should be provided.

While all the above was going on in SG1, another busy group was Library Evolution Working Group (LEWG). They were working their way through a long list of papers. Something that caught my eye on Wednesday afternoon was 'Towards a Transaction-safe C++ Standard Library: `std::list`' (N3862). This discussion involved the work of a study group (SG5, Transactional Memory) moving forward into a working group (LEWG). This paper documents the efforts of SG5 in using `std::list` to work towards a transactional STL, beginning by applying the transactional memory support in GCC 4.9 to `std::list`. SG5 are in the process of preparing a transactional memory TS based on the contents of the paper 'Transactional Memory Support for C++' (N3919). A straw poll was taken on whether their should be a library part to the TS, the result being eleven in favour with none against.

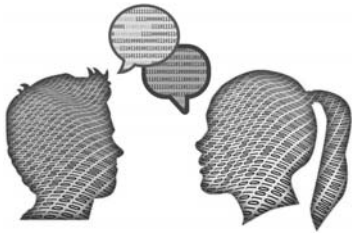
Another LEWG discussion that caught my eye was the discussion of the paper 'Contiguous Iterators: A Refinement of Random Access Iterators' (N3884). The reason this caught my eye goes back to Dublin in 1999, my first C++ Standards Committee meeting at ISO level. At that meeting a much discussed topic in the Library Working Group (LWG) was the question of whether or not `std::vector` was contiguous in memory. The question had been asked a lot (on the news groups) by developers interfacing to C code, who wanted to know if they could use a `std::vector` as an array and pass it to C code. That is, they wanted to be able to write expressions of the form `f(&v[0])` and be assured such code would work. Obviously the answer was: yes, `std::vector` is contiguous in memory. However, the reason for all the discussion was that it turns out to be very hard to say this, in any satisfactory way, in the standard. Why this is so, is beyond the scope of this article (I've digressed quite a lot already). Suffice to say that the correct form of words was found by the next ISO meeting. However, you can now see why a paper on adding a contiguous iterator to the classifications of iterator would catch my eye. There are other containers that are obviously contiguous in memory, `std::string` for example, and N3884 proposes to generalise what was done for `std::vector` by putting the contiguous memory guarantee in the iterator classification. Reading the minutes of this meeting, it appears that there was some scepticism regarding this approach. Another approach based on traits was suggested, and found favour in a straw poll. However, the minutes are not sufficiently detailed for me to figure out the details. This is another thing I will endeavour to find out by the time I write my next report.

MARK RADFORD

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

Code Critique Competition 87

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



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

Last issue's code

> This code works for me but not for my co-worker. I get
>
> `31-Mar-2013 01:30:00`
>
> and they get:
>
> `java.text.ParseException: Unparseable date: "2013-03-31T01:30:00-04:00"`
>
> What's wrong with their machine? I can't easily check as they're in a different country!"

Can you solve the mystery and identify any other reasons why the code might be problematic?

The code is in Listing 1.

Listing 1

```
import java.text.*;
import java.util.*;
public class DateTimeTest
{
  static String format =
    "yyyy-MM-dd'T'HH:mm:ss-SSS";
  public static void main(String[] args)
  {
    SimpleDateFormat sdf =
      new SimpleDateFormat();
    sdf.applyPattern(format);
    sdf.setLenient(false); // Require 2-digits
    testParse(sdf);
  }
  static void testParse(SimpleDateFormat sdf)
  {
    Date then = new Date();
    try
    {
      then = sdf.parse(
        "2013-03-31T01:30:00-04:00");
    }
    catch (Exception ex)
    {
      System.out.println(ex);
      System.exit(1);
    }
    DateFormat df =
      DateFormat.getDateTimeInstance();
    System.out.println(df.format(then));
  }
}
```

Critiques

Russel Winder <russel@winder.org.uk>

A few general comments first:

Star imports are really anathema in Java, they should be banned. The reason is the pollution of namespace that happens using them. By being explicit about each and every import, it acts as documentation, and there is no pollution of the name space.

We will leave the incorrect positioning of the open braces as a bikeshed discussion.

Far too many unnecessary line breaks in the code.

There are far too few `final` keywords in this code: far too many variables. Proper use of `final` to create single assignment variables is an element of defensive programming that leads to less errors and hence fewer debugging sessions.

The catch block catches `Exception`, this is far too broad a catch. Catch blocks should catch only the exceptions being handled in the catch block. In this case `ParseException`.

The then variable has to be initialized but why to an actual `Date` instance. It turns out this is a deeper issue than at first sight as `null` is a poor initializer in a Java context where fluent interfaces assume non-null references. More on this shortly.

Now on to the bug itself:

The problem lies in the format string: `-SSS` is not a time zone specifier but a very weird format for the milliseconds of an instant. The date to be parsed is full ISO 8601 compliant with time zone but not fractional seconds. The immediate solution is to replace `-SSS` with `X` (note no - since it might be a +).

This raises the question of "How did this ever work?" for the original poster. The only solution I can come up with for this is that original poster added the `sdf.setLenient(false)` after trying things out without rechecking. Without the enforced lack of leniency, the parser spots the ISO 8601 format and does the right thing despite the format specifier. Remove leniency and it can never have parsed.

So the updated code:

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.Date;
```

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

Standards Report (continued)

In my previous report I talked about SG1 discussing the paper 'A proposal to rename `shared_mutex` to `shared_timed_mutex`' (N3891). At the time I wrote that SG1 were in favour, but it needed the approval of the full committee. Just to tie up this loose end, let me finish by mentioning that, at the end of week, a motion was submitted to full committee and was carried.

```
public class RW_DateTimeCheck {
  static String format =
    "yyyy-MM-dd'T'HH:mm:ssX";
  public static void main(final String[] args) {
    final SimpleDateFormat sdf =
      new SimpleDateFormat();
    sdf.applyPattern(format);
    sdf.setLenient(false); // Require 2-digits
    checkParse(sdf);
  }
  static void checkParse(
    final SimpleDateFormat sdf) {
    Date then = null;  // Have to initialize and
   // update to avoid uninitialized check at (1)
    try {
      then = sdf.parse(
        "2013-03-31T01:30:00-04:00");
    }
    catch (ParseException ex) {
      System.out.println(ex);
      System.exit(1);
    }
    final DateFormat df
      = DateFormat.getDateTimeInstance();
    System.out.println(df.format(then));  // (1)
  }
}
```

works as required everywhere. Notice that the names have been changed to remove the [Tt]est. This is not a test program, it is a quick check of something. So do not use test here, it gives the wrong impression to the reader.

Whilst on testing, the original poster has written a program that is nigh on untestable at the unit level. It can still be system tested of course, but should it be reworked so that **checkParse** can be unit tested? The answer is yes, but we leave it as an exercise to actually do this.

Of course the above is the wrong answer. Now that we have Java 8 with JSR310, we have a much better system of dates and times. The real answer is therefore:

```
import java.time.ZonedDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeParseException;
class RW_ThreeTenVersion {
  public static void main(final String[] args) {
    ZonedDateTime then = null;  // Have to
    // initialize and update to avoid
    // uninitialized check at (1)
    try {
      then = ZonedDateTime.parse(
        "2013-03-31T01:30:00-04:00");
    }
    catch (DateTimeParseException dtpe) {
      System.out.println(dtpe);
      System.exit(1);
    }
    System.out.println(then.withZoneSameInstant(
      ZoneId.systemDefault()));  // (1)
  }
}
```

well except that the point about unit testing is as valid for this program as for the previous ones. This is left as an exercise for the reader. Along with deleting all non-JSR310 date/time code and replacing it with JSR310 variants. Check out **java.time** and use only classes found therein, or used therein.

Commentary

This code demonstrates several misunderstandings over the use of Java's original data time methods; I think Russel covered these fairly comprehensively.

However, the code originally worked quite happily for some users and not for others for an unrelated reason: the failure is actually not caused by the erroneous use of **-SSS** to try and match the timezone -04:00.

The actual problem is that the time specified in *some* timezones falls during the hour when the clocks go forward at the start of daylight savings. So if, for instance, you run the program using a timezone of Europe/London then the system tries to interpret the date – ignoring the supplied timezone – in local time and fails as it is invalid: there *was* no such local time as the clocks skipped from 01:00 to 02:00. Setting a UTC timezone allows the time to parse.

For a user with, for example, an American timezone setting this time is valid (as America entered daylight saving earlier in the year) and for a user with, for example, a Japanese timezone the date is valid as Japan does not use daylight savings. This is the reason why the code worked for the questioner but not for his colleague in a *different* country (and timezone).

Unfortunately the root cause of the failure to parse the time value is not made clear in the error message associated with the exception.

Equally unfortunately, the original code does not display any timezone information; this might have helped to uncover the difference in settings that was causing the different behaviour being experienced by the two programmers.

As Russel correctly observes, Java now provides an improved way of handling dates and times; however, it is still important for the programmer to understand the actual timezone requirements of the processing that is required.

A second problem with the code is that the (poorly named) **testParse** method calls **System.exit**. This should be strongly discouraged as it is a bad habit – deciding to terminate the application is rarely a decision that can be made *inside* a function and, as Russel noted, it makes the code extremely hard to test.

The third problem with the way the exception handling is written is that the local variable **then** must be given a value since the variable is used beyond the **try** … **catch** block. Unfortunately, although the reader of the code knows that the catch block is fatal, the language rules require that the variable is initialised in case an exception occurs.

Refactoring the function to throw an exception (or return) in the **catch** block avoids needing to give an initial value to the variable. (It is hard to make a good choice – a newly constructed date gives the risk of introducing a spurious value into the program on a parse failure, whereas **null** is a poor choice since it is all too easy to cause an unwanted **NullPointerException**.)

The Winner of CC 86

This critique only attracted one entry; and while Russel did provide a good critique of many of the problems with the code I don't think he actually answered the original question of why the code worked for one user and not for another; so I am not awarding the prize this time round. Perhaps the issue with the date provided was a bit too subtle for our readers?

Code Critique 87

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

> I was adding items into a map only if they weren't already there and the code reviewer said I shouldn't be using operator[] - see how I was doing it in the function called **old_way**. So I tried to do it better using the code in **new_way**. Now the code sometimes works and sometimes doesn't – can you help?

The code is in Listing 2 (next page).

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.

Bookcase

The latest roundup of book reviews.

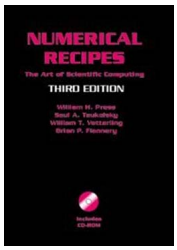
If you want to review a book, your first port of call should be the members section of the ACCU website, which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

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

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

Jez Higgins (jez@jezuk.co.uk)

Numerical Recipes Third Edition: The Art of Scientific Programming



William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, ISBN: 978-0-521-88407-5, published 2007, 1235 pages

Reviewed by Paul Floyd

This is a book that I ought to have read decades ago. The thing that put me off was the very negative comments on usenet and mailing lists about the quality of the C++ example code. More on that in a bit. Perhaps stung by these remarks, there is a put-down of the weak attempts at numerical analysis often found in computer science texts.

I've worked in or around numerical analysis for most of my career, so about 1/3 of the material in this book was reasonably familiar to me.

The positive points are that the book offers a lot of example code and is quite compendious, so if you just want to jump in and get something working, it's a great resource. Beware though that just buying the book doesn't get you access to the code (unless you're prepared to type it yourself). I bought book + CD, otherwise you have to pay extra to access the source. If you want to redistribute anything compiled with *Numerical Recipes* source, that is also licensed. The licensing is fairly draconian, and also something of a bone of contention, being used as another reason to avoid the book.

I have seen some of the code from the 2nd edition, and the 3rd edition is considerably better. It's still not great, certainly not production quality code. Gone are the 1-based Fortran arrays. It is a great shame that the code integrates poorly with standard library containers – they have their own Vector and Matrix template classes which have interfaces that are incompatible with the standard library algorithms. Some of the code is just ugly. I'll just give two examples. The `Base_interp` class contains two pointers to `Doub` which need to point to arrays with lifetimes at least as long as the `Base_interp` instance. For reasons that escape me, the constructor takes a pointer and reference to Vector array. One of the member pointers is initialised with the constructor formal argument, the other member pointer with `&x[0]`. Secondly, the Hash class has a 'special tweak' in order to be able to handle C-strings. This is to 'special case' elements of size 1 and to treat them as C-strings. I reckon that this could be done cleanly with the right template specialization. It would be interesting to compare the performance with some `std::unsorted_map` implementations.

Despite these critical remarks, the breadth of the coverage is a great strength, and for this I give quite a high rating.

Code Critique Competition 87 (continued)

Listing 2

```
#include <iostream>
#include <iterator>
#include <map>
#include <string>
std::map<int, std::string> theMap;
void old_way(int key, std::string value)
{
  if (theMap[key].empty())
  {
    theMap[key] = value;
  }
}
void new_way(int key, std::string value)
{
  std::map<int, std::string>::iterator it
    = theMap.lower_bound(key);
  if (it->first < key)
  {
    theMap.insert(it,
      std::map<int, std::string>::value_type
      (key, value));
  }
}
```

Listing 2 (cont'd)

```
namespace std
{
  ostream& operator<<(ostream&,
     map<int, string>::value_type const &rhs)
  {
    return cout << rhs.first
      << "=" << rhs.second;
  }
}
int main()
{
  new_way(1, "test");
  new_way(2, "another");
  new_way(1, "ignored");
  std::copy(theMap.begin(), theMap.end(),
   std::ostream_iterator<
    std::map<int, std::string>::value_type>
    (std::cout, "\n"));
}
```

View from without the Chair

Roger Orr
publications@accu.org

We held the Annual General Meeting (AGM) at the Marriott Hotel in Bristol on the Saturday of the conference.

The formal minutes from the outgoing secretary will appear in due course; but we wanted to provide a summary of the meeting for those unable to attend in person.

This was the first year where electronic voting has been available and, to the best of our knowledge, this resulted in more people casting votes at this AGM than at any previous AGM of the association.

There were a few small teething problems but overall the system appeared to be a success.

The draft accounts were presented and the good news is that we have (again) made a surplus over course of the year, so this is encouraging for the organisation.

We elected the following members onto the committee for 2014/15:

Executive members

- Matthew Jones (Membership Secretary and Local Groups)
- Robert Pauer (Treasurer)

Non-executive members

- Roger Orr (Publications)
- Seb Rose (Advertising)
- Chris O'Dell (Study Groups and Social Media)
- Mark Radford (Standards)
- Dirk Haun

As previously mentioned however, we did not have any candidates proposed by the deadline for the post of Chair and Secretary.

We did not have a formal proposal for a caretaker post holder at the meeting. Hence the new committee can continue the normal running of the association but must also arrange for a Special General Meeting (SGM) for the election of a new Chair and Secretary (see below).

We voted in favour of the motion to discontinue the 'hardship fund' and to roll this money into general finances. The fund has been called on much less in recent years and an earmarked fund no longer seemed necessary. The committee can exercise their discretion in cases of hardship without requiring a specific fund.

Thanks are due to those who are standing down from the committee for their work over the previous year:

- Alan Griffiths (Chair)
- Giovanni Asproni (Secretary)
- Mick Brooks (Membership Secretary)
- Astrid Byro (Publicity)
- Tom Hughes
- Andrew Marlow (Co-opted)

First committee meeting

We held our first committee meeting later that afternoon to discuss the options going forward. A number of ACCU members were also in attendance. The minutes will be circulated to accu-members once approved.

We noted that Alan Lenton has put his name forward as a candidate for the role of Chair, and we also have a (tentative) offer from Malcolm Noyes who is considering the role of secretary.

At the meeting we co-opted Alan and Malcolm, and also Ralph McArdell who offered his assistance, to the committee. We are grateful to all three for their involvement.

The next committee meeting is planned for 4th May – and so should have happened by the time this issue of *CVu* is printed and dispatched.

At this meeting we plan to pick the date for the Special General Meeting at which we hope to elect the two missing officers. There is a long lead-time for this because the timing changes introduced at last year's AGM to support electronic voting mean the same timescale for SGMs as for AGMs.

Special General Meeting

The special general meeting will be to elect a new Chair and Secretary. We will be offering electronic voting and are expecting most – if not all – of the members of ACCU will use this method for registering their vote.

We would like to have a choice of candidates for both roles, so if you think you could be a candidate for the role of Chair or Secretary please find two members of ACCU to propose and second you and send your name and their confirmation of support to secretary@accu.org. Alternatively if you think someone else might be suited to the role please suggest that they might like to put themselves forward.

There is a description of the roles (and the other committee roles) on the members section of the ACCU website – visible once you log on to the site.

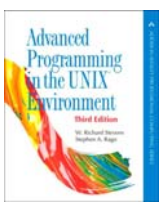
Future tasks for the committee

One matter of concern that was raised by the membership secretary was a slow decline in membership; over the last year membership has dropped by 38 to 649 and there was a similar sort of drop the year before. We are trying to collect more information from the members who do not renew to provide us with data on the reason, or reasons, behind this decline.

While this is not yet a critical problem it is important that the committee (and the incoming Chair) focus on this issue and investigate whether, for example, it is time to make changes to the way the association runs to reflect the many changes in the world of IT, the Internet and social media since the association began.

Bookcase (continued)

Advanced Programming in the UNIX Environment, Third Edition



By W. Richard Stevens, Stephen A. Rago, ISBN: 978-0-321-63773-4, published 2013, 1024 pages

Reviewed by Paul Floyd

Another 'third edition'. I likes the first edition so much that I've also bought both the 2nd and 3rd editions. However, I've downgraded my rating a bit. This isn't so much a reflection on any lowering in the quality of the book as an indication that I feel that the relative quality of available information on the internet has improved.

This edition is updated to cover more recent UNIX (and clone) systems and standards. Whilst the versions of Linux used was reasonably up to date when the book was being written (presumably 2012), I was a bit disappointed to see that the versions of FreeBSD (8.2), Solaris (10) and Mac OS X (10.6.8) were already a bit old. Standards-wise, there are a lot of updates to cover the Single UNIX Specification, version 4 (SUSv4). The main changes that I noted were the removal of STREAMS (though Rago does seem rather nostalgic and mentions it in passing several times) and more of a move in style from purely Stevens to more Rago, e.g., adding more larger code examples with comments.