zieren +++ Berauschend: Schweizer Informatikunternehmen betreibt Alkoholismusförderung im VIS +++ Gesucht: Bilokale Kaffeemasch ung an zw

VISIONEN

www.visionen.ethz.ch

September 2010

Aufbau

Magazin des Vereins der Informatik Studierenden an der ETH Zürich (VIS)

mission control™
security services

Become a Mission Control Security Service Engineer and join us for a journey around the world.

Well-known companies, institutions and NGOs secure the availability of their crucial IT- and communications infrastructure with Mission Control Security Services in over 100 countries. Our team is constantly looking for new technically trained employees who have a solid background in computer science, and experience in Internet technologies. We offer you advanced-level internal development training, enabling you to become a certified Mission Control Security Service Engineer, working in a global, fast-paced and highly dynamic environment in our Operation Centers in Zürich and – if you like to – Sydney, Australia. Please join us on this journey around the world. **www.open.ch**

Mission Control Security Services by Open Systems AG

Editorial

FABIAN HAHN — AUFBAU AUCH ALS HOBBY?

Informatik an der ETH ist ein Ingenieurstudiengang. Wer schon etwas Studienzeit auf dem Buckel hat, dem sollte dies bereits während dem Semester mit praktischen Programmierübungen und -projekten bewusst geworden sein. Den neueingetretenen Erstsemestrigen, von welchen die meisten mit dieser Ausgabe wohl zum ersten Mal ein Visionen in den Händen halten, wird es spätestens bis zum Abschluss des Basisjahrs auch so ergehen. So viel Mühe sich die ETH jedoch mit ihrer wohldurchdachten Ausbildung geben mag, uns das theoretische Hintergrundwissen näherzubringen und die mathematischen Grundlagen unseres Berufs zu verstehen – die Praxis stellt uns jeweils vor ganz andere Probleme. Über kurz oder lang wird jeder feststellen, dass man sich Programmierung nicht durch Vorlesungen sondern nur durch viel Übung aneignen kann, schliesslich handelt es sich um ein Handwerk wie jedes andere auch. Die eigentliche Kunst besteht dann darin, einerseits vorausschauend zu planen, und andererseits die gelernte Theorie stets im Hinterkopf zu behalten, um sie im richten Moment zur Anwendung bringen zu können. Auch dies erfordert hartes Training.

Umso näher liegt es auf der Hand, diese Fähigkeiten nicht nur während Übungsserien an der ETH auszubauen, sondern sich auch privat mit der Thematik auseinanderzusetzen, teils aus Spass, teils bewusst mit dem Ziel, weitere Erfahrungen zu sammeln. Programmieren als Hobby ist nicht nur äusserst ergiebig (schliesslich kann man sich damit tolle Sachen zur eigenen Verwendung basteln!), sondern auch äusserst interessant, da es einem die Möglichkeit gibt, einmal über den Tellerrand unseres Studiumstoffes zu blicken und zu erfahren, wie es sich anfühlt, wenn man einmal kein säuberlich vorbereitetes Framework für die Lösung einer Serie vom Assistenten erhält, was sich ja sonst mehr wie das Ausfüllen eines Lückentexts als wie wirkliche "Entwicklung" anfühlt.

Bevor ich euch in der nächsten Ausgabe verraten werde, womit ich mir selbst in dieser Beziehung meine Freizeit fülle, würde es mich brennend interesseren, wofür unsere geneigten Leser in ihrer Freizeit ihre Programmierfähigkeiten einsetzen. Je nach Feedback werde ich mir auch einige Projekte genauer ansehen und im Visionen vorstellen. Meldet euch doch einfach bei hahn@vis.ethz.ch und erzählt mir von euren Entwicklungen!

Euer Chefredakteur,
Fabian Hahn

Inhalt

Aufbau

Berichte

Studium

Spass

avaloq

essential for banking

Wir kreieren Bankensoftware von morgen – steigen Sie noch heute bei uns ein

Sie sind in der modernen ICT-Welt zuhause und interessieren sich für die spannenden internationalen Finanzmärkte? Wollen Sie Ihr Wissen in einem dynamischen und professionellen Umfeld einbringen? Dann sind Sie bei uns genau richtig!

Wir suchen:

– Software Engineers
– Business Analysten
– Projektleiter
– Sales- und Business Consultants

Weitere aktuelle Stellenangebote finden Sie unter: www.jobs.avaloq.com
Wir freuen uns auf Ihre Online-Bewerbung auf careers@avaloq.com
Unser HR-Team beantwortet gerne Ihre Fragen unter +41 58 316 10 10

Die Avaloq Gruppe ist Schweizer Marktführer für integrierte Bankensoftware. Mit dem Avaloq Banking System hat das Unternehmen eine integrierte und modular einsetzbare Gesamtbankenlösung für Privat-, Retail- und Universalbanken, Vermögensverwalter sowie Transaktionsbanken auf dem Markt, auf die bereits 44 Kunden mit über 65 Banken in mehr als 300 Zweigstellen und Niederlassungen weltweit vertrauen. Daneben bietet Avaloq Serviceleistungen über den gesamten Lebenszyklus der Banking Software an. In der Avaloq Academy werden Kunden zur selbständigen Weiterentwicklung des Avaloq Banking Systems ausgebildet. Die Avaloq Community ist für über 35'000 User von Avaloq Plattform zum Austausch von Know-how und Innovationen. Im Zuge der Internationalisierungsstrategie werden auch im Ausland weitere Niederlassungen angestrebt.

Zurich
Geneva
Frankfurt
Luxembourg
London
Vienna
Moscow
Dubai
Singapore
Hong Kong

UNIVERSUM®
TOP 100
IDEAL EMPLOYER
2010 STUDENT SURVEY

www.avaloq.com

Begrüssung

VON PROF. FRIEDEMANN MATTERN

Liebe Studentinnen und Studenten des ersten Semesters!

Als Vorsteher des Departements Informatik darf ich Sie, auch im Namen aller Departementsmitglieder, zum Bachelor-Studiengang im Herbstsemester 2010 herzlich begrüssen! Ich freue mich sehr, dass mit Ihnen wieder viele junge Menschen den Weg zu uns gefunden haben!

Mit dem Studium der Informatik an der ETH Zürich haben Sie eine Ihre persönliche Zukunft prägende Wahl getroffen, zu der man Ihnen in doppelter Hinsicht gratulieren darf: Zum einen studieren Sie nun an einer der weltweit renommiertesten Universitäten, und zum anderen gehört das Fach Informatik zu den entscheidenden Gebieten, die unsere Zukunft (und zwar nicht nur meine und Ihre, sondern die der ganzen Menschheit!) wesentlich beeinflussen. Doch dazu später mehr.

Zunächst aber: Was erwartet Sie nun in den nächsten Jahren? Natürlich viele neue und manchmal auch ganz fundamentale Einsichten und Erkenntnisse rund um die Informatik als Wissenschaft, spannende Studienprojekte und ein grossartiges Angebot, Ihre Fähigkeiten und praktischen Kenntnisse zu erweitern und zu vertiefen. Viele Menschen, die nicht diese Chance haben, beneiden Sie um diese Möglichkeiten!

Aber vor den Erfolg haben die Götter bekanntlich den Schweiss gesetzt. Auch das darf nicht verschwiegen werden: Das Studium ist, insbesondere in den ersten Semestern, nicht nur anstrengend, sondern sehr anstrengend! Sie werden durch viele „gleichzeitige" Lehrveranstaltungen stark gefordert, das Niveau ist hoch, es geht schnell voran und Ihre Mitstudierenden („Kommilitonen", sagte man zu meiner Zeit noch, also „Mitkämpfer") sind auch nicht von gestern. Vor allem aber werden Sie am Anfang des Studiums (wann denn auch sonst!) intensiv die Grundlagen des Faches studieren, und diese sind keineswegs immer „spassig", sondern oft mathematisch und fast immer theoretisch. Praxis und Theorie sind aber keine Gegensätze, beides gehört zusammen, und beides wird an der ETH auch betrieben. Der Informatiker Gregor Snelting hat es einmal so ausgedrückt: „Theorie ohne Praxis ist steril, Praxis ohne Theorie ist unfruchtbar". Und tatsächlich gibt es oft nichts Praktischeres als eine gute Theorie! Dass es bei einem universitären Studium oft theoretisch und abstrakt zugeht, hat noch einen anderen Grund: Ein hohes Abstraktionsniveau zu erlangen, ist zwar mühsam – aber es macht eben auch frei: Vom höheren Standpunkt aus sieht man klarer und versteht die Zusammenhänge viel besser, kann selbstbewusster Entscheidungen fällen und gewinnt tiefe Einsichten, die dann sogar zu einer echten Leidenschaft und Freude am Fach heranwachsen können.

Doch zurück zur ETH, die für Sie nun viele Stunden in der Woche die geistige und physische Umgebung darstellt. Sie ist gross und wirkt anfangs undurchschaubar. (Auch im ganz wörtlichen Sinne: Im Hauptgebäude kann man sich auch nach Monaten noch verlaufen!) Wer kann Orientierung bieten? Zunächst einmal die „älteren Semester", die ja schon alles durchgemacht haben und Erfahrungen gesammelt haben. Bei den Vertretern des VIS, dem Verein der Informatik-Studierenden, finden Sie die Ansprechpersonen. Aber scheuen Sie sich auch nicht, die Professoren und Assistenten anzusprechen, insbesondere dann, wenn Sie fachlichen Rat benötigen. Zwar geht es am Anfang in den grösseren Vorlesungen eher unpersönlich zu, aber gerade deswegen freuen sich die Professoren (nun ja, die allermeisten jedenfalls), auch mit Einzelnen einmal zu sprechen – und seien es auch nur ein paar Minuten nach der Vorlesung oder in den Pausen.

Orientierung zum Fach und Anregung zugleich bieten auch gute Lehrbücher und Fachzeitschriften – und zwar aufgrund des strengen wissenschaftlichen Begutachtungsprozesses noch oft in wesentlich besserer Qualität als das, was man im Internet zu einem Fachthema findet. Die Informatik-Bibliothek hält diese vor und lädt zum Stöbern ein. Nutzen Sie dieses reichhaltige Angebot, es kann Ihren Horizont über das hinaus, was Sie in den Vorlesungen hören, wesentlich erweitern und durch eine andere Perspektive auch manches verständlicher machen. An dieser Stelle nur ein einziger Tipp zu einem echten Klassiker unter den vielen Lehrbüchern: „The Art of Computer Programming" von Donald Knuth in (bislang) drei Bänden. Zwar ist die Darstellung manchmal ein bisschen exzentrisch (Donald Knuth, übrigens ein Ehrendoktor unseres Departements, ist es schliesslich auch), aber wie Knuth selbst sind die Bücher einfach genial! Zu den Fachzeitschriften ein ganz anderer Tipp: Das „Informatik-Spektrum" erscheint etwa alle zwei Monate und enthält fachlich hochstehende, aber nicht allzu abgehobene und auf Deutsch geschriebene Artikel zu relevanten und aktuellen Themen der Informatik. (Mitglieder der Schweizer Informatik Gesellschaft oder der deutschen Gesellschaft für Informatik erhalten diese Zeitschrift übrigens umsonst.) Mit der Zeit erhält man damit einen guten Überblick zu praktisch allen wichtigen Themen. Das gilt in gleicher Weise auch für die sehr empfehlenswerte englischsprachige Zeitschrift „Communications of the ACM".

Womit wir wieder bei der Informatik wären. Warum ist diese so spannend? Einerseits natürlich deshalb, weil sich das zwischen den Ingenieur- und den Naturwissenschaften angesiedelte Fach zusammen mit seinen Anwendungen so rasant weiterentwickelt. Eine fast grenzenlose Verfügbarkeit von Information „immer und überall", Suchmaschinen, elektronische Bücher, eigene Videos „im Netz", soziale Netzwerke – alles das war vor nur 25 Jahren schliesslich noch Science-Fiction. Andererseits aber, weil gerade etwas Einzigartiges geschieht: Die Informatik durchdringt, ähnlich wie früher die Mathematik, viele andere Wissenschaften und dringt nun sogar auch in die physische Welt, und damit in fast alle Lebensbereiche, ein. Internet und PC sind zwar schon jetzt allgegenwärtig, doch das alleine ist nicht entscheidend. Viel bedeutender ist, dass die Informatik in Form kleinster Prozessoren, ausgestattet mit komplexer Software, unsere physische Welt, und damit unseren Alltag auch jenseits von PC und Internet, immer weiter erobert – ein schleichender Prozess, den

➔

die Öffentlichkeit kaum explizit wahrnimmt, obschon er bereits vor einigen Jahren begonnen hat. Denn tatsächlich interagieren wir bereits heute, im Zeitalter von Mobiltelefonen, computergesteuerten Haushaltsgeräten und digitaler Unterhaltungselektronik, täglich unbewusst mit Hunderten von kleinen Computern, auf denen oft komplexe Informatik-Systeme implementiert sind: Wann immer wir Auto fahren, Wäsche waschen, Kaffee kochen, Aufzüge verwenden, Musik hören oder telefonieren, nutzen wir dabei verborgene Computersysteme, die uns diese Tätigkeiten bequemer und sicherer erledigen lassen als früher. Der Fortschritt von Mikroelektronik und Informatik geht aber ungebremst weiter, so dass unsere Welt schon bald durchsetzt sein dürfte von praktisch unsichtbaren Computersystemen, die mit Sensoren ihre Umgebung laufend erfassen und die aktuelle Situation interpretieren, um dann miteinander zu kooperieren und mittels Aktoren steuernd in die Realität einzugreifen – um uns und die natürlichen Ressourcen zu schützen, und um unser Leben noch interessanter und anregender zu gestalten.

Was die ständig fortschreitende Informatisierung der „realen" Welt letztendlich wirklich bedeutet, das kann man heute noch kaum abschätzen. Die Wirkungen dürften auf lange Sicht aber gewaltig sein. Um diesen Prozess, der uns alle betrifft, gut zu meistern, dazu bedarf es vor allem breit, verantwortungsvoll und fachlich exzellent ausgebildete Informatik-Experten. Mehr als 30 Professorinnen und Professoren aus über 10 Nationen stellen sich in unserem Departement dieser Aufgabe. Darüber hinaus werden Sie in Tutorien und Praktika auch viele engagierte Assistenten und Assistentinnen kennenlernen, und eher im Hintergrund halten viele Angestellte den ganzen Betrieb am Laufen. Und doch ist unser Departement nur ein Teil des Ganzen, jenseits der Informatik bietet die ETH auf hohem Niveau noch viel mehr. Auch wenn Sie durch Ihr Studium sehr gefordert werden – nutzen Sie, zumindest gelegentlich, das vielfältige ETH-Angebot in Wissenschaft, Weiterbildung und Sport. Es lohnt sich!

Ich wünsche Ihnen zum Studienbeginn, aber vor allem auch für die nun folgenden Jahre an der ETH, viel Erfolg. Daneben auch viel Freude ausserhalb des Studiums in einer neuen Lebensphase, die von Herausforderungen geprägt und keineswegs immer einfach ist, die man weitgehend eigenständig meistern muss, in der man aber auch vielfältige Erfahrungen sammelt, neue Freunde kennenlernt und in der man die Weichen für sein späteres Leben stellt. Denn es gibt für Sie ja ein wirklich chancenreiches Leben nach dem Studium; und auf dieses, zumindest was den Beruf betrifft, wird die ETH Sie nun umfassend und nach dem neuesten Stand der Wissenschaft vorbereiten.

Fühlen Sie sich willkommen bei uns – ich freue mich darauf, viele von Ihnen im Laufe der Zeit persönlich kennenzulernen!

Prof. Friedemann Mattern
Vorsteher des Departements Informatik

LOCATION: ZURICH

ONE YOU
One Credit Suisse

ALEXANDER WOLLTE MEHR VERANTWORTUNG. WIR VERTRAUTEN IHM UNSERE WICHTIGSTEN KUNDEN AN. Als Alexander mit der Betreuung seiner neuen Private Banking Kunden startete, war nicht abzusehen, wohin ihn sein Engagement führen würde. Heute berät er 50 unserer grössten Kunden in Nahost. Seine Empfehlungen sind für die Kunden sehr wertvoll – die Erfahrung für ihn unbezahlbar. Lesen Sie Alexanders Geschichte unter credit-suisse.com/careers

CREDIT SUISSE

valgrind

FABIAN HAHN — CODE GRINDER

Like programming in C or C++ in Linux? Then you won't get around using a good memory debugger such as valgrind to keep your code clean.

Low-level languages such as C and C++ are often said to be unreliable and error-prone to work with. In fact, they aren't even low-level languages, they're just lower-level than the Java, C# or Eiffel you are used to working with. One of the many reasons that earned them this reputation is the lack of a garbage collection mechanism, the fact that you yourself as a programmer are responsible for all memory allocated and used by your program. Allocating, for the most part, isn't that difficult and rarely the cause of any problems. However, being responsible for your memory also means you need to free it after using it. Simply forgetting about it won't help—if you lose your last pointer to an allocated piece of memory, it will live on for the rest of your program's execution time. In that case, the memory is said to be leaking.

Careful software engineering and design of your program will of course help reduce such mistakes, but you won't be able to prevent them all from happening. In all cases, you will leak some memory and you will write into unallocated space at some point, no matter how careful you are. In C and C++, these issues are simply another kind of bug that may occur, and as you are well aware, you can never prevent all bugs while writing software.

Now let's say you wrote an application and notice that you must have introduced some kind of memory bug. Mostly, they appear as random crashes during execution (since the kernel kills them with a SIGSEGV signal, or segmentation faults, as they like to be called), as garbled console output (random ASCII noise when you would expect alphanumeric characters) or as a monotonic increase in memory consumption for long-running processes. How do you find that bug?

memcheck – a memory checker

That's exactly where *valgrind* comes in. In it's basic form, *valgrind* acts as a memory checker that controls all kinds of access to your RAM by interpreting your compiled code in some kind of x86 emulator. The virtual machine is able to detect any read or write access to an unallocated memory area, invalid frees or deletes, branches depending on uninitialized variables and leaked memory at the end of a program's execution. All of this useful magic can be achieved by simply prepending "valgrind" to the beginning of your program as you would execute it in a shell. In case you added debug symbols to your binary (using gcc's "-g" flag), you will even see the exact locations in the code where the

problem occurred. Let us for example consider the following C++ program which contains an obvious out-of-bounds memory access as well as a memory leak:

```
int main()
{
        int *array = new int[10];
        for(int i = 0; i <= 10; i++) {
                array[i] = i;
        }
return 0;
```

First, the out of bounds access occurs as soon as the loop variable i reaches the value 10 and second, the memory that array points to is never freed. This is what valgrind spits out if we run "valgrind ./simple":

```
==4599== Invalid write of size 4
==4599==    at 0x8048516: main (simple.cpp:5)
==4599==  Address 0x42cb050 is 0 bytes after a block of size 40 alloc›d
==4599==    at 0x4026F2F: operator new[](unsigned int) (in /usr/lib/valgrind/  …
            vgpreload_memcheck-x86-linux.so)
==4599==    by 0x80484F8: main (simple.cpp:3)
==4599== HEAP SUMMARY:
==4599==     in use at exit: 40 bytes in 1 blocks
==4599==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==4599== LEAK SUMMARY:
==4599==    definitely lost: 40 bytes in 1 blocks
==4599==    indirectly lost: 0 bytes in 0 blocks
==4599==      possibly lost: 0 bytes in 0 blocks
==4599==    still reachable: 0 bytes in 0 blocks
==4599==         suppressed: 0 bytes in 0 blocks
==4599== Rerun with --leak-check=full to see details of leaked memory
```

Let's parse this together! The first thing *valgrind* notices is an "invalid write of size 4" at line 5 of our simple program. Unsurprisingly, this happens to be our out of bounds access to our allocated memory slice. It even tells us at which point the invalid write access happens: "0 bytes after a block of size 40 alloc'd". Considering that sizeof(int) == 4, and 4 * 10 = 40, this is exactly our off-by-one mistake when the i counter reaches 10. On the next line of the first report, we even get a stack trace of the allocation call for the memory block to which *valgrind* suspects the access had been intended. In that case, this was the new operator from the C++ standard library that we used to request our buffer space.

Second, we get an overview of the situation after the program execution. Again, *valgrind* notices that there was "1 allocs", but only ➔

"0 frees", which can't be a good thing. The following leak summary tells us more about it: "40 bytes in 1 blocks" are "definitely lost", which is undoubtedly our array buffer we never released using the delete operator. If we value *valgrind*'s advice and rerun the whole thing with the "--leak-check=full" option, we even get a stack trace of of the initialization of our leaked memory block:

```
==4651== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4651==    at 0x4026F2F: operator new[](unsigned int) (in /usr/lib/    …
            valgrind/vgpreload_memcheck-x86-linux.so)
==4651==    by 0x80484F8: main (simple.cpp:3)
```

There it is – *valgrind* couldn't find a matching delete for the new on line 3.

There's even more

Already amazed with what *valgrind* can do for you? Well guess what, this was just the tip of the iceberg. It turns out *valgrind* isn't just a memory checking tool, but actually a collection of tools among which the memory checker, *memcheck* for short, was only one. *Cachegrind*, another of the included tools, is a full-fledged cache and branch-prediction profiler. Now what does that mean? In basic terms, it allows you to see what you normally don't see if you run a program directly on your processor: The number of cache hits and misses, the corresponding levels in the memory hierarchy (L1, L2, worse…) and the number of branches for which their outcome was correctly predicted. Cache misses are something you generally want to get rid off as far as possible since going to a deeper level in the memory costs CPU cycles (about 10 for an L1 miss and up to 200 for an L2 miss on usual processors) and forces your program to wait, which results in a performance loss. The same goes for branch prediction: Since the execution of individual instructions in a processor is pipelined, it often has to predict the execution path the program will be taking. For example, the body of an if statement in C++ might get executed before the processor actually knows that the condition leading to the execution of that code is satisfied. If it finds out that it is, everything is okay and execution may continue. If not, there is a problem—everything up to the beginning of the conditional block needs to be reverted, and that too costs valuable CPU cycles. *Cachegrind* helps you track the efficiency of your program in those two aspects by producing statistics about both of them, which is especially helpful for performance-critical sections of your code.

The third tool included in *valgrind* I'd like to mention here is *callgrind*. *Callgrind* is very similar to other classical profiling tools such as *gprof*, with two vast differences. On one hand, it is a lot slower which is due to the overhead of running the program in an emulator and on the other hand, it actually works. If you ever used *gprof*, you might remember it as producing useless output, outputting nothing at all or simply fai- ➔

CREDIT SUISSE

Adrenalin, Fun, Genuss!

Einfach Konto eröffnen und unvergessliches Erlebnis aussuchen.

SICHERE DIR EINS VON ÜBER 60 ERLEBNISSEN. JETZT!

Lass dich mit einer Massage im Spa oder mit schmackhaften Gaumenfreuden verwöhnen. Flieg selber einmal eine Cessna. Versuch dich als Schokoladentester oder erleb River Rafting vom Feinsten. Übernachte in einem traumhaften Hotel in idyllischer Umgebung oder nimm an einer Rallye teil. Eröffne einfach dein kostenloses Privatkonto Academica und schon kannst du aus über 60 Erlebnissen genau das richtige für dich auswählen. Mehr Informationen findest du unter **www.credit-suisse.com/erlebnisse**.

Auszug aus den Bedingungen: Das Angebot ist limitiert und gilt daher nur so lange Vorrat. Nur ein Gutschein pro Kunde. Ausschliesslich für Academica-Neukunden. Weitere Angebote und Informationen zur Erlebnis-Box sowie die vollständigen Bedingungen findest du unter www.credit-suisse.com/erlebnisse.

ling as soon as you use a shared library – it might work for your fibonacci number calculator, but for real-word tasks, it's just a big pain. Things look differently for *callgrind*, since once you get over the ages of execution time you have to endure to complete a run, it actually produces very useful output and exactly what you'd expect from a profiler. Things get even better if you use the marvelous graphical *KCachegrid* tool to interpret your results (yes, I polluted my system with KDE libraries just for this one!). You will notice that the notion of a call graph becomes completely different if you can actually see it. It allows you to identify the most costly regions of your code and enables you to exactly put your finger on bottlenecks that are worth eliminating. As you might have guessed already by the name of this tool, it was originally designed for the output of *cachegrind* and hence also works with *cachegrind* results.

Conclusion

Did I mention there is still more? *Helgrind* and *DRD* to debug threads, *Massif* to profile heap usage and *ptrcheck* to find array overruns even *memcheck* is unable to find. All together, in my opinion the *valgrind* suite with all its individual tools is an essential helper in developing good C/C++ software in Linux. I even caught myself once porting Windows code to Linux for the sole purpose of running *valgrind* on it when Visual Studio failed to help me out (yes, *memcheck* found the bug afterwards). If you are interested in the topic or in using the tools yourself in more depth, I highly recommend *valgrind*'s excellent documentation which makes a great read right before going to bed at night. Not only does it describe the functionality of its tools, but it also features common pitfalls and background information on its design and on how the results are produced.

A great side effect of using *valgrind* is that you automatically try to keep your code clean. If you're not happy with it until *valgrind* is too, it will save you numerous hours of hunting after memory bugs later and will definitely reduce the number of users of your software complaining about "strange crashes" after you deploy it. Furthermore, since buffer overflows often bear security risks, you have killed two birds with one stone with very little effort. Programming in C and C++ surely has its drawbacks, but being afraid of memory bugs is no reason not to use them for performance-critical applications. After all, there are tools to help you and so you should use them. Happy grinding! ◆

Compiler Design—A Lecture Review

SIMON GERBER — LIKES LOW-LEVEL STUFF

A lecture for everyone who likes to program and wants to know more about compilers.

The Compiler Design lecture (formerly Compiler Design I) concentrates on the basic principles of a compiler, namely parsing the source code, building a symbol table, semantic checking, and simple code generation. Advanced topics for optimising and JIT compilers are covered in the Compiler Design II lecture which hopefully will take place again next spring.

In the exercises (of Compiler Design I), teams of two students each implemented a simple compiler for a subset of the Java language called *Javali*[1] in Java. There was an exercise for each part of the compiler, and the exercises were built on top of a framework which actually lets you compile small programs written in *Javali* and run them. As *Javali* is object-oriented and has simple inheritance, it also shows how object orientation works—using vtables and the like—on a low level. The compiler itself produces assembly code for x86 (or any other architecture) which is then converted to object code by an assembler for the chosen architecture.

The lecture itself provides information on the theoretical aspects of compiler design, such as formal grammars and code analysis, in addition to looking at the topics needed for the exercises.

As the lecture is now part of the Compulsory Major Courses of the computer science bachelor program, apparently the grading and possibly also the exercises have changed. The grade for the old course (Compiler Design I) was comprised of a 1 hour exam which counted for 1/3 of the grade and the exercises which counted for 2/3 of the grade. According to the Course Catalogue, the new lecture will have a written examination with a duration of 3 hours[2]. Unfortunately there is no information available yet on how the exercises for the new course will look.

Nevertheless anyone who is interested in the principles behind a compiler should consider taking the course. ◆

Links

[1] http://www.lst.inf.ethz.ch/teaching/lectures/ss09/222/assignments/javali.pdf

[2] http://www.vvz.ethz.ch/Vorlesungsverzeichnis/lerneinheitPre.do?lerneinheitId=69216&semkez=2010W&lang=en

The Theory of a Balanced Challenge

RUDOLF MAXIMILIAN SCHREIER —PROFESSIONAL RAMBLER

Probably humanity's second most favorite, and certainly second earliest practiced hobby is the concept of a game. But why do humans enjoy games, and how can one ensure that each challenge a player has to face is both fair and enthralling at the same time?

Disclaimer: I am not a philosopher or a psychologist. Everything below is only a theory that I came up with, and should not be considered life advice.

The Illusion of Happiness

First of all, one has to understand what it means to have fun. For the sake of this article, let's assume that fun equals the illusion of happiness. So what a game has to achieve is both make each player forget their troubles and allow them to achieve something which makes them happy, i.e. in the broadest sense "success". Given the large differences between individuals, this is quite the undertaking; a few key points however always remain the same: A player is handed tools with which to reach a goal, according to a specific set of rules. As this holds true also of real life, perhaps "winning" in a game of this structure equals "winning" real life, i.e. finding happiness.

Learning (by Trial and Error)

In both real life and games, a player starts out with no experience, and mostly vague instructions and guidelines. So like a child learns from its parents, a player has to be taught the necessary skills; in the best case, from his or her own past mistakes, as in the classical principle of Trial and Error. The problem with this scenario lies in the fact that while a human lifetime lasts many decades, a game will probably not, so having players make mistakes and spend time learning from them will not make it fun.

The Learning Curve

So usually, a player is presented with select simple challenges to complete with their limited knowledge, upon which each skill is then based and furthered. The level of skills required throughout a game and the ease with which those skills are learned is commonly called the "Learning Curve". This curve has been an obstac-

le in many a game's path to commercial success, as it clearly separates the customers into the categories of hardcore and casual players; those that are willing to invest much time and effort into furthering their success, and those that are not. Finding the right balance here is crucial to the point that it is highly common to find games with an adjustable degree of difficulty.

Difficulty and Cheating

And while in many cases, raising difficulty levels might only result in numerical changes such as number and health of enemies, often a high level of difficulty results in gameplay changes, and possibly a more realistic game experience. But wait – wasn't the whole point of a game to forget about reality for a moment? Of course, however, it is quite against human nature to accept success for a challenge which demanded too little effort; and for many people, features such as a limited AI or Auto-Aim add the feeling of "playing dirty", i.e. cheating the challenge. But again, everyone will feel different about this. Therefore, there have been video games with over 5 gradient levels of difficulty, to allow for crassly varied adventures.

Playing To Your Strengths

The common point of gameplay to all games is using your own advantages to trump someone else's disadvantages. This may be the simple stone-paper-scissors mechanism, which is found in real-time strategy games to this day, but is still very exploitable: A very recent example is Valve's *Team Fortress 2*. The 9 classes of the tactical multiplayer shooter can be grouped into 3 groups of Attack, Defense and Support, or, alternatively, into triples of running speed: Fast, Medium, and

➔

After his third defeat this week, King Edmundo started to seriously question the balancing of Chess.

Delicious fruit veils a deceptive learning curve.

Slow. And just as well that stone-paper-scissors can be adapted to varieties with an odd number of materials , as every class has its distinct boons and banes.

Staying On Top

And the main reason why *TF2* and nearly all titles of famed Blizzard Entertainment enjoy an incredibly long and healthy life, is that their balancing is insurmountable, and stays that way while new elements are introduced. As an example, take Blizzard's *Diablo II*. Published in 2000, it has received only one commercial add-on in 2001. However, its latest patch numbers 1.13 and was released in only March of 2010. That is no typo; a whole ten years after the original release, features are added and balanced to reward the players which remain true to the franchise.

The Proof is in The Pudding

So for all intents and purposes, a correct balance consists of at least these three pillars: One, a consistent learning curve to suit both casual and hardcore players. Two, gameplay that requires and rewards individuality, catering to strengths and weaknesses. Three, keeping your eyes wide open, be it simple tinkering with floating point numbers to achieve true fairness, or giving characters a wider set of tools at their side. These are the very foundation of everything which wants to call itself a balanced game; and to be straightforward: a game without balance is not a game, it is a farce. After all, what holds true of real life and love should also hold true of an abstraction of them: Even if you don't know what the end holds in store for you, The path is the goal. ◆

Balancing II — Victory Conditions

STEFAN THÖNI — IS IN A STATE OF EQUILIBRIUM

I've seen many well-conceived games fail because they just weren't any fun to play. Not that they lacked action or had too many bugs, but they simply had bad victory conditions. Either the game continued forever because the victory condition was never satisfied or it was satisfied before decisive action could even begin. But games are most fun to play while victory is imminent but the victor is undetermined.

The Setting

For a deeper investigation into victory conditions we've got to narrow the discussion a little: In the following I'm writing about competitive multiplayer strategy games. While campaigns provide a good learning curve for new players and even an odd hour of fun for enthusiasts, multiplayer games, especially against other humans, definitely have a longer half-life. I love turn-based strategy games but a multiplayer match that continues over several dozen hours of playtime is just too difficult to arrange. Therefore I'll limit the following discussion to real-time strategy games.

Traditionalism

By tradition, real-time strategy games provide an annihilation or destroy-all-enemies mode with only one victory condition: kill all enemies. In my opinion this is the worst victory condition possible because after the first decisive engagement one side is likely to gain the upper hand for the remainder of the game. Worse, the remainder may take a long time compared to the decisive phase. Add to that the utter frustration of the loser, after every single one of his units has been slaughtered and the last of his buildings razed. All of this is even more true if part of the game is harvesting resources. Not only does this add a buildup phase in the beginning were almost no combat takes place, but it also increases the slippery-slope effect by allowing the victor of the initial engagement to hamper the underdog's collection of resources and therefor diminish his chances further.

A Brave New World

Some game designers have begun to notice that other victory conditions might make their games more fun to play. One of these is the con- ➔

cept of taking and holding specific locations on the map leading to a drain in the opponent's tickets. The game is therefore lost when a player's tickets reach zero. This kind of victory condition can provide a prolonged, more interesting fight for these positions. Most games however combine this victory condition with the gathering of resources and base building. This opens up the possibility of a base rush, an all-out attack on the enemy headquarter leading to an early end of the game. Moreover such a base rush is often possible by avoiding the enemy entrenched at the victory position and finding his base only lightly defended. This can be countered by letting players build powerful defenses at their bases or even better have these prebuilt at the start of the game.

Looking around

Looking at other genres one can see that alternatives have already been developed. For example, some shooters have a so-called stopwatch mode, in which the game is divided into two phases: With one team attacking and the other defending, an objective must be reached, after which the roles are switched. The winner is determined by the team that was faster in attacking. In my opinion this mode would also work great for real-time strategy games. It would force decisive action from the beginning to the last second of play. Additionally it would be more diverse than traditional game modes and allow different map layouts considered unfair in symmetric game modes. To make it more interesting one could imagine giving the defending team an initial advantage in resources or units and the attacking team an ever increasing stream of resources or units, guaranteeing that the attackers would eventually overwhelm the defenders. But which team can hold out longer as defender or attack faster with less resources?

Take-home message

I think all game designers should carefully consider game modes and victory conditions. An interesting game mode can't make a badly designed game good, but it can make a good game great. And a lousy victory condition can certainly break an otherwise good game. As implementing a different game mode or victory condition is very cheap compared to creating an engine and all the artwork, games should feature more alternative modes or at least add more of them in addons. I hope to see more creativity from game designers in this area in the future. ◆

High-Tech am Zürichsee

C#, Python, agile Entwicklung und UML sind wichtige Schlüssel für unser starkes Wachstum. Starte deine Karriere in einer professionellen und dynamischen Umgebung.

www.sensirion.com

SENSIRION
THE SENSOR COMPANY

The Birth of a Debian Package

ANDREAS BRAUCHLI — PRICE INCLUDES PACKAGING

The job of a packaging sytem is clear: keeping track of what dirt lies where on the system and removing it without traces once its time has come. So we are here today to witness the birth of a Debian package.

Hobby Packaging

For the hobby packagers of you who just envy gentooers for having the source to almost any package and do not intend to redistribute their binary packages but still are reluctant to just "`make install`" any package into a dark corner of the system—well possibly until the next HDD failure—can use a handy tool called *checkinstall* ("`sudo apt-get install checkinstall`").

That is if the package is not already in the huge Debian/Ubuntu repository (search with "`apt-cache search pkgname`") and the source tarball (file ending in .tar.gz) does not come with a debian folder for easy building.

If the package already has a debian folder you'll just have to run "`fakeroot debian/rules binary`" to generate the .deb and "`dpkg -i ../pkgname_vers.deb`" to install it.

Checkinstall

Checkinstall is a handy tool for keeping track of installed files and generating a simple debian package that will remove all of these upon deletion. (Yeah I don't trust "`make uninstall`" either)

It works with just one additional command:

```
/configure && make && sudo …
checkinstall make install
```

Checkinstall will allow you to optionally specify a few variables such as the name and revision of the package used, as well as its binary dependencies. Even though optional, it is of course recommended to set as much information as possible. Also note that obviously the packages listed as dependencies must be installed on your system at the time of running *checkinstall* or the installation will fail and result in a broken package (can be fixed by running "`apt-get -f install`" which will just remove the broken package).

Great Package Deal

For the real package deal it's a bit more complicated but also more rewarding if you get to be the maintainer of a package that possibly millions of people will be using… but here's how the story goes:

The Gathering of the Tools

First get your toolbox together with:

```
sudo apt-get install ...
build-essential devscripts ...
ubuntu-dev-tools debhelper ...
dh-make diff patch cdbs quilt...
gnupg fakeroot lintian ...
pbuilder piuparts
```

(obviously there is no *ubuntu-dev-tools* in Debian, that's for the Ubuntu folks)

Create the *pbuilder* environment (to build packages in isolation)

```
sudo pbuilder create
```

If you already have one it might be time to update it again. Run

```
sudo pbuilder update
```

If any of your new packages has dependencies on a package that can only be found on Ubuntu's universe or multiverse repository you need to add the following line to /etc/pbuilderrc:

```
COMPONENTS="main restricted ...
universe multiverse"
```

If you are not building for other architectures than your own you may also add this line:

```
APTCACHE=/var/cache/apt/ ...
archives
```

to not re-download packages that you already have installed.

Debianize the World!

Get original source of the software you wish to debianize. Get it either as tarball if possible or make one by not specifying -f to dh_make in one of the next steps.

Extract the source and change the directory name to the format "pkgname-0.10.1".

A note on terminology: pkgname will always refer to the name of your package in this article. It will sometimes be libpkgname if the package in question is a library.

Creating the Debian Folder Skeleton

Enter the source directory and run:

```
dh_make -e ...
debian-geek@student.ethz.ch ...
-f ../pkgname-src.orig.tar.gz
```

Additional optional parameters are:

- `-c license` (one of: gpl, gpl2, gpl3, lgpl, lgpl2 lgpl3, artistic, apache or bsd)
- `-p pkgname` or `libpkgname` when building libraries

We'll choose to build a package with *cdbs* by entering [b] at the package type prompt. Overlook the changes and confirm or abort with [CTRL]+[C] to correct.

The debian folder has now been created inside the source folder, move into it ("`cd debian`"). The real work is only about to start:

Example files

There are a few example files (ending in *.ex / *.EX) we'll just be using watch.ex by renaming it to "watch" ("`mv watch.ex watch`") and delete the others ("`rm *.ex *.EX`").

Watch out

The watch file will contain just following lines. Note that Sourceforge projects can use the simplified URL:

```
version=3
http://sf.net/opende/ ...
ode-(.+)\.tar\.gz
```

Now running "`uscan`" in the source directory will check if newer versions of the software might be available.

➔

Misc Files

Edit the changelog file (the Debian package revision number will be taken from the top most change).

Edit the copyright file to reflect the license of the package.

Delete or modify README.Debian and README.source.

If creating multiple packages from a single source (like -dev -dbg -doc etc.) create pkgname-foo.install files for each package with the file masks of what goes where. E.g. libpkgname-dev.install would contain:

```
usr/include/pkgname/*
usr/lib/libpkgname*.*a
usr/share/pkgconfig/*
```

Don't forget the create additional sections for each package in the control file in the next step (but have just one source paragraph).

You're in control

Changing the control file is crucial for a good package as it holds all the meta informations for *dpkg*, the debian packaging system.

We'll go through a few fields founds in the control file. The ones found in the skeleton but left out here should be obvious.

Bump the Standards-Version field to whatever is up to date (3.9.1 as of writing).

The Sections field could be libs, devel or much more (Get the section of your favorite Debian package with "`apt-cache show bzflag | grep Section`").

If a control file has a source, libpkgname and libpkgname-dev paragraph, the sections will be libs, libs and libdevel.

Priority will probably be optional (known not to conflict with anything) or extra (could possibly create a conflict with other packages) —I mostly use extra.

The Depends field will indicate package dependencies of the resulting binary packages. Note that a libpkg-dev package should depend on its corresponding library of the same version "`libpkg (= ${binary:Version})`".

If you have a pkgname-doc section, your binary should list it as suggested dependency: "`Suggest: pkgname-doc (=${binary:Version})`".

The Build-Depends field of the first (source) paragraph contains the development headers required to build the binary packages.

All package related fields can have modifiers (=,<=,>=,<<,>>) to indicate that at least a certain version of a package is required.

The Architecture will be "any" (should build on all architectures) or "all" (architecture-independent stuff like documentation which is portable without rebuilding).

The Description field constitutes of a short description header (60 chars) on the same line as the field followed by a more lengthy description indented by a single space on each of the following lines. Paragraphs can be made with a single indented dot ("."") on a line.

```
Package: libpkgname-dev
Description: libpkgname is a cool library
(development files)
This awesome library includes files
.
This package contains the development files for
libpkgname
```

debian/rules

Now the rules file—which thanks to *cdbs* isn't that complex anymore. Make sure to define modifier variables (if needed) to the included rules or classes before including them!

If your source uses *cmake* as build system the

modifier flags will be of the form:

```
DEB_CMAKE_EXTRA_FLAGS := ...
-DFOO_FEATURE="BOOL:ON"
include /usr/share/cdbs/1/ ...
class/cmake.mk
```

If you need to patch the source of the original package (e.g. your changes haven't been accepted upstream yet or are Debian/Ubuntu specific changes that aren't accepted) the easiest is to create a debian/patch folder and place your patches (with file ending in .patch and a strip-level of preferrably 1, though 0 or 2 will also work) in there. Then including simple-patchsys.mk:

```
include /usr/share/cdbs/1/ ...
rules/simple-patchsys.mk
```

If you don't have patch files, they can be made with

```
diff -ruN                  ...
pkgname-version.orig       ...
pkgname-version >          ...
pkgname-version/debian/    ...
patches/fix-foo.patch
```

on the original pristine folder and the folder with the changes applied.

It's best to name the patch files on what they do (e.g. fix-cmake-foo-module.patch) and to have multiple logically grouped patch files instead of one big changeset.

Building the Source Package

Once this is all done we can build the source package (which is not really one package, but a collection of files) by entering "`debuild -S -sa`" in the source folder. Have a look at the output of *lintian* which checks for common packaging misstakes. You will also be requested to enter the GPG key for the provided email address to sign the source package.

Building the Binary Package

Now we're ready to build the real binary package. We'll do it isolated from the current system installation to make sure that the listed dependencies match to what the software actually requires. This means that building will fail if you have not set the Build-Depends field correctly in the control file even if they are otherwise installed on your system.

Start the process with:

```
sudo pbuilder build ...
../pkgname*.dsc
```

The resulting binaries will be written to /var/cache/pbuilder/result/ upon success.

Good luck!

Install the package(s) with "`dpkg -i /var/cache/pbuilder/result/pkgname*.deb`"

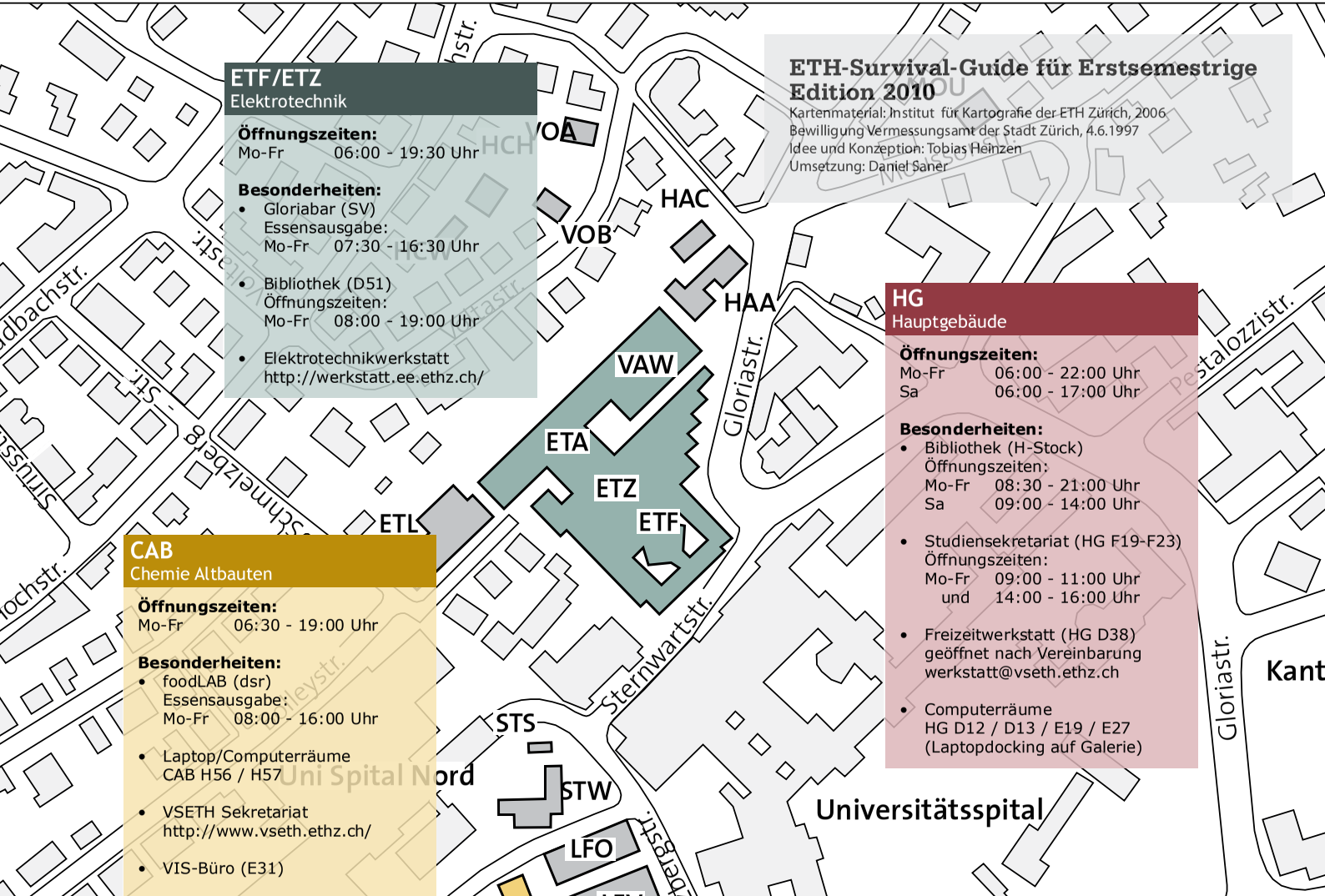
What's left

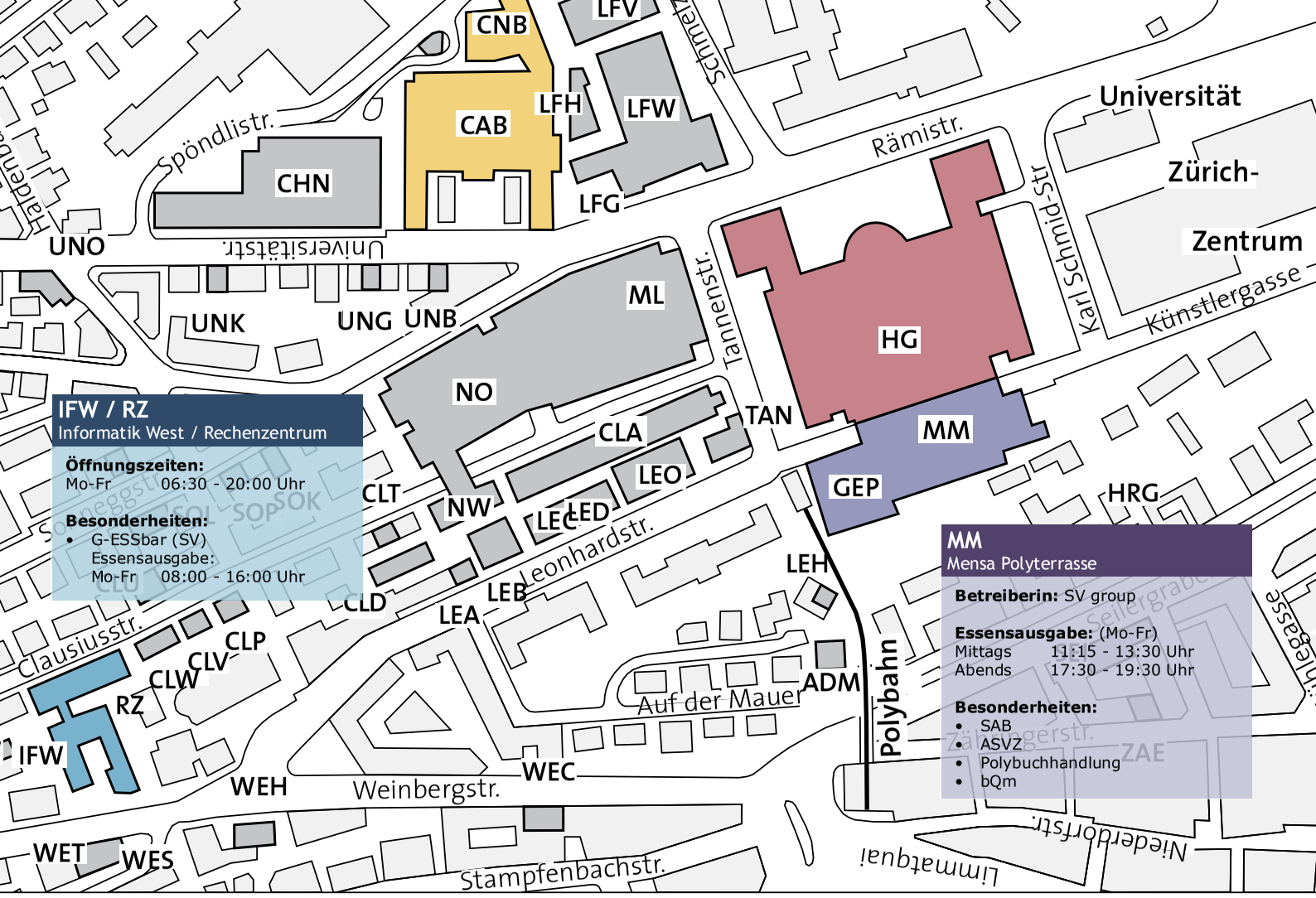
The remaining steps will be to distribute the package on Ubunt''s PPA, another Debian repository or simply on your website.

Note that *pbuilder* can also be used to build packages against i386 on an amd64 machine. If uploaded there, the debian build server will probably try to build it against all its architectures which might result in a few frustrating surprises depending on how portable the software is that you're trying to build. ◆

Additional information on packaging can be found on:

https://wiki.ubuntu.com/PackagingGuide
https://wiki.ubuntu.com/PackagingGuide/Complete
https://wiki.ubuntu.com/PbuilderHowto
http://wiki.debian.org/PkgSplit
http://www.willnichols.me.uk/progs/debpack/
irc://irc.oftc.net/#debian-mentors


ETF/ETZ
Elektrotechnik
Öffnungszeiten:
Mo-Fr 06:00 - 19:30 Uhr
Besonderheiten:
• Gloriabar (SV)
Essensausgabe:
Mo-Fr 07:30 - 16:30 Uhr
• Bibliothek (D51)
Öffnungszeiten:
Mo-Fr 08:00 - 19:00 Uhr
• Elektrotechnikwerkstatt
http://werkstatt.ee.ethz.ch/
ETH-Survival-Guide für Erstsemestrige
Edition 2010
Kartenmaterial: Institut für Kartografie der ETH Zürich, 2006
Bewilligung Vermessungsamt der Stadt Zürich, 4.6.1997
Idee und Konzeption: Tobias Heinzen
Umsetzung: Daniel Saner
HG
Hauptgebäude
Öffnungszeiten:
Mo-Fr 06:00 - 22:00 Uhr
Sa 06:00 - 17:00 Uhr
Besonderheiten:
• Bibliothek (H-Stock)
Öffnungszeiten:
Mo-Fr 08:30 - 21:00 Uhr
Sa 09:00 - 14:00 Uhr
• Studiensekretariat (HG F19-F23)
Öffnungszeiten:
Mo-Fr 09:00 - 11:00 Uhr
und 14:00 - 16:00 Uhr
• Freizeitwerkstatt (HG D38)
geöffnet nach Vereinbarung
werkstatt@vseth.ethz.ch
• Computerräume
HG D12 / D13 / E19 / E27
(Laptopdocking auf Galerie)
CAB
Chemie Altbauten
Öffnungszeiten:
Mo-Fr 06:30 - 19:00 Uhr
Besonderheiten:
• foodLAB (dsr)
Essensausgabe:
Mo-Fr 08:00 - 16:00 Uhr
• Laptop/Computerräume
CAB H56 / H57
• VSETH Sekretariat
http://www.vseth.ethz.ch/
• VIS-Büro (E31)
HCH
HCW
VOA
VOB
HAC
HAA
VAW
ETA
ETZ
ETF
ETL
STS
STW
LFO
Gloriastr.
Sternwartstr.
Universitätsspital
Uni Spital Nord
Kant



IFW / RZ
Informatik West / Rechenzentrum

Öffnungszeiten:
Mo-Fr 06:30 - 20:00 Uhr

Besonderheiten:
- G-ESSbar (SV)
  Essensausgabe:
  Mo-Fr 08:00 - 16:00 Uhr

MM
Mensa Polyterrasse

Betreiberin: SV group

Essensausgabe: (Mo-Fr)
Mittags 11:15 - 13:30 Uhr
Abends 17:30 - 19:30 Uhr

Besonderheiten:
- SAB
- ASVZ
- Polybuchhandlung
- bQm

Das Partytagebuch

VON LUKAS HUMBEL, MARC BRUGGMANN UND REMO GISI

Bevor wir wieder ins Partysemester starten wollen wir versuchen, uns ans letzte Semester zu erinnern. Im Selbsttest erkundeten die Autoren letzten Frühling unzählige Studentenparties und wollen nun hier davon berichten.

Achtung!
Folgende Schilderungen basieren auf prahlerischen Augenzeugenberichten, vernebelten Erinnerungen, hemmungslosen Verleumdungen, ärztlichen Rapporten, offensichtlichen Unwahrheiten und anderen Räubergeschichten. Genuss auf eigene Gefahr.

Woche 1

Warmup – Klassiker ★★★★☆

Die vom Challenge organisierte Party gehört zu den absoluten Klassikern. Heiss und eng, dunkel, laut und alkoholisch. So muss eine StuZ-Party aussehen. Es empfiehlt sich, früh da zu sein.

ESN Welcome Party – Randgruppen ★★★★☆

Wie jedes Jahr brilliert die Willkommensfeier der Austauschstudenten mit aussergewöhnlichem Ansturm auf die Bar und füllt wieder einmal den grossen Saal des Dynamos. Zur Abwechslung nur wenige Kleidungsstücke geklaut, kaum Personen verloren und brav um 6 Uhr morgens – mit oder ohne Austauschstudentin – im Bett.

Woche 2

French Kiss – Randgruppen ★★★☆☆

Das allseits bekannte Phänomen, dass lokale Kulturvereine super Parties schmeissen, hat sich wieder mal bestätigt. Die welschen Studis hauen kräftig auf den Putz!

Woche 3

Greenhouse Dance: Balkan Party – Kostümparty 

Wie es sich für eine Mischmasch-Party-Location gehört, hat auch eine "Balkan Beats Party" ihren Platz. Gut gelauntes Partyvolk amüsiert sich zu balkanischer Strassenmusik. Leider war die Fete nach dem Konzert bald vorbei.

Woche 4

International StuZ – Randgruppen 

Typische ESN-Party mit durchschnittlicher Besetzung. Boobies from everywhere!

Exzess im Labor – Kostümparty 

Passionierten StuZpartygängern sicher ein Begriff. Gratis-Eintritt mit Labormantel, welcher mit einem hübschen Stempel verziert wird. Nicht nur für Labormantelbesitzer ein Muss!

Woche 5

Medibar – Chicks 

Medizinerinnen im StuZ? Tatsächlich verlaufen sich ab und an auch Unistudentinnen an die ETH! Tolle Aussichten und auch sonst unübliches Publikum.

LM Party: Hot & Spicy – std::party 

Scharfe Schoten locken den Jüngling ins Partylokal, Schützengarten sorgt dafür dass er bleibt. Eine willkommene Abwechslung in der Turbine. Die Regulars tanzen auf der Bühne. Nice.

VIS Poker Night – Games 

Knapp 100 Casino-Begeisterte pilgern mit Anzug und Sonnenbrille bewaffnet in den StuZ, um den Pokerkönig zu krönen. Tolle Deko mit Selbstbeweihräucherung.

Griechenparty – Randgruppen 

Der griechische Verein säuft sich ins Koma. Drinks normalerweise halbe-halbe, plus noch etwas über die Hände des besoffenen Barkeepers. Wer noch stehen kann tanzt händewedelnd zu griechischer Discomusik.

➔

Woche 8

Pharma-Party – Chicks 

Lächerliche Happy-Hour mit drei Franken pro Bier. Verwirrung. Rum-Colas um einiges stärker als der DJ. Signifikant weniger Instanzen des weiblichen Geschlechts als erwartet, Gay Overdrive in Barnähe.

Woche 9

Lake it easy – Altersheimausflug 

Der VSETH lädt zur Rundfahrt auf dem Zürichsee. Der Filz schlürft teures Heineken aus der Dose, die Brodwurscht ist grusig. Eine handvoll Leute partysiert noch ein bisschen an der Afterparty im StuZ. Sack ist ein toller DJ.

Woche 10

VPP: Vorstandspizzaplausch – Intern 

Studenten aller Fachvereine, bewegt eure Ärsche in den Vorstand. Es lohnt sich nur schon für diese Party. Obwohl vielleicht nur der VIS den allgemein Promillepegel so hochpeitschen kann.

Sexual Selection – std::party 

Dem Traditionsanlass geht langsam der Schnauf aus. Wenigstens gibts Hot-Dogs, serviert vom VIS-Präsidenten.

Woche 11

The Nerd Party – Kostümparty 

VCS und VIS laden zum Streberbesäufnis. Super Motto, geile Party, Scheiss-DJ. GUV-Chicks im Schulmädchenoutfit hinter der Bar. Auch der Rollstuhl-Transportservice ins Unispital wird fleissig genutzt.

Woche 12

Conquering Afterparty – Klassiker 

Es wird gemunkelt, dass der Bierverbrauch pro Person über fünf Liter betragen haben soll. Da kann man auch mit so einer Party nichts mehr falsch machen. Ausser sich einen (VIS-)Bären aufzubinden. Der Präsi läuft mit blutiger Nase zum Spital, der Rest macht weiter Party.

Spontanparty im Gang – Intern 

Teilweise noch verkatert vom Conquering treffen sich acht Leute im Gang. Zehn Stunden Party, >= 16 Runden Shots, Schnupftabak und Falkenbier. BlockRocker sorgt für Stimmung. Alle nackt um 6 Uhr, frittierte Bratwürste um 7 Uhr.

Woche 13

Your Professor Rocks – Konzert 

Super-Konzert von "Not the Sensational Alex Harvey Band", nur war leider kaum jemand da. Schande übers OK.

Woche 14

Lux en Boum – Randgruppen 

Die Luxemburger feiern ohne bestimmten Grund, aber trotzdem gut. Wer es fertig bringt, seine zwei Bier auf Luxemburgisch zu bestellen, kann von der frühen Happy Hour profitieren – und trinkt dann vier Bier.

Fazit

Auch wenn die Parties immer dann am besten sind, wenn man sie selbst organisiert: Feiern im StuZ lohnt sich allemal. In diesem Sinne: Viel Spass im kommenden Semester! ◆

Upcoming Fall Pilots

BY RUDOLF MAXIMILIAN SCHREIER — WATCHES OUT FOR IT SO YOU ALSO HAVE TO. WAIT…

New fall term, new fall TV season! An exemplary outlook at a few of the new series which American TV stations are throwing into the ring.

In contrast to spring and summer, the shows starting in fall are mostly new licenses, each offering a pilot episode and first season; while there are certainly many interesting returning shows, I'll strictly focus on the truly new ideas, to allow a fresh start for newcomers. For those also interested in recurring series, feel free to visit the Futon Critic's Showatch[1] for an overview of past, current, and upcoming TV material.

$#*! My Dad Says

What, really, is the point of taking a Social Networking phenomenon and adapting it to a television medium? I completely missed efforts such as *lonelygirl15*, and now CBS have taken on a truly mind-boggling effort: Transforming 140-character tweets into 21-minute TV episodes.

Justin Halpern's twitter[2] contains all of the gems of life that his father Sam shares with him, mostly containing cusses and political incorrectness that seems impossible to transfer to broadcastable material. So naturally, what CBS have done is invent a background story that embellishes the characters' interaction into a soap-y series, and censoring the title to the above combination of special characters, which are supposed to be pronounced "Bleep My Dad Says". At this point, I personally could not imagine this ever working out for more than a mini-series. The only thing that could save this effort from complete mediocrity is good writing and casting. And while the pilot was co-written by Halpern himself, the episodes themselves will rest on the shoulders of dedicated writing personnel. So, who could act as the centerpiece of a comedy based around an eccentric, swearing 72-year-old? If you thought William Shatner, you had the same stroke of genius as CBS, and are perhaps familiar with his character in *Boston Legal*.

So while the source material is brilliantly hilarious, and the writing is sure to be quite rocky, perhaps the acting can save this show from utter failure. Episodes run Thursdays, starting September 23th, on CBS.

Skins (US)

For those of you used to the USA's unfortunate custom of buying up the licenses of successful European and especially UK-made shows, you may not be surprised to hear that *Skins* (UK), which had seen its premiere in 2007, and since then has been called "the most authentic teen brand on TV" has met the same fate, having been picked up by MTV, of all things. *Skins* being a teenager slice-of-life show, I am not going to mention why this is not a brilliant idea, and shall merely describe the most attractive features of the original series, and let you as the reader imagine what MTV and American TV culture (and I use this word very grudgingly) will probably do to them:

First of all, as a person for whom English is very close to their heart, the most endearing trait that springs to mind when watching Skins is its unbridled use of the British accent; probably chosen for the enhanced air of authenticity, it makes or breaks the show for each and every viewer.

Secondly, the will to provoke and uproar. While this may seem a very intellectual thing to like about a series, it really lays the foundation for the very earthy feel of a show that is all about the realism. And if you've been through your teenage years and did not spend them learning yet another Perl-inspired dynamic programming language, you know that the basis of teenage reality is gratuitous profanity, nudity, substance abuse and raw, soul-shattering emotion. At this point, I remind you that the BBC have allowed the use of the F-word after the 10pm-watershed, while the USA, in turn, have what the makers of *Family Guy* have rightfully called "the freaking FCC".

All in all, I will be very surprised if this turns out at all different from *The O.C.*, a lifeless husk of a show which only serves the purpose to keep its audience dumbed down, and buying the next best single that the music industry pushes out like clockwork. Be it un- or fortunately, MTV have delayed start of the series to the first quarter of 2011. ◆

[1] http://www.thefutoncritic.com/showatch/
[2] http://twitter.com/shitmydadsays

Die wunderbare Welt der Programmierwettbewerbe

VON SEBASTIAN MILLIUS UND SANDRO FEUZ

Du programmierst nicht nur gerne, um am Schluss eine benutzbare Applikation abzuliefern, sondern auch, um dich mit anderen Gleichgesinnten zu messen? Dann stellen wir dir hier die wichtigsten Programmierwettbewerbe vor, schliesslich ist manchmal auch der Weg das Ziel.

Der Franzose Pierre de Cubertin wurde am 1. Januar 1863 in Paris geboren und gilt als der Begründer der Olypischen Spiele der Neuzeit. Er erkannte den Sport als eine Möglichkeit, Menschen aller Länder zusammenzubringen. Sich mit anderen messen zu können ist im Wesen des Menschen wohl ebenso tief eingewurzelt wie der Drang, Geheimnisse aufzudecken. Programmierwettbewerbe bieten beides.

In einem Programmierwettbewerb werden Aufgaben gestellt, die mittels eines Computerprogrammes zu lösen sind. Dabei liegt der Fokus auf algorithmischem Programmieren, das heisst es müssen keine grafischen Oberflächen o.ä. implementiert werden, sondern nur Programme, welche die korrekte Ausgabe zu den verfügbaren Eingaben generieren. Programmierwettbewerbe bieten denjenigen eine Plattform, denen das universitäre Angebot nicht den Lernhunger zu stillen vermag oder die bereit sind, unbekannte Gebiete zu erforschen und neue herausfordernde Konzepte zu lernen.

Wie immer steht dabei vorallem der Spass im Vordergrund.

ACM ICPC

Für Studenten der wichtigste Wettbewerb ist der ACM ICPC, der International Collegiate Programming Contest der Association for Computing Machinery. Es ist ein internationaler Wettbewerb, an dem Studierende von Universitäten der ganzen Welt teilnehmen um sich zu messen.

Die 1947 gegründete ACM ist die älteste und grösste Informatikervereinigung der Welt. Die Organisation mit Hauptsitz in New York vergibt unter anderem den Turing Award (das Äquivalent zur Fields-Medaille der Mathematiker oder dem Nobelpreis der Wissenschaftler) und fördert die Fähigkeiten von Studenten der Informatik. Aus diesem Grundgedanken heraus entstand auch der ICPC.

Der ICPC ist ein Team-Wettbewerb und wird in drei Phasen abgehalten. Die erlaubten Programmiersprachen sind C, C++ und Java.

1. **Local Contest:** alle ETH-Studenten, welche die Teilnahmebedingungen erfüllen (studiert seit 2006 oder später, geboren '87 oder später. Genauer auf [1]), können am lokalen

Wettbewerb am 9. Oktober hier an der ETH teilnehmen. Der Wettbewerb dauert 5 Stunden und besteht aus 5 bis 10 Problemen. Dabei ist jeder auf sich alleine gestellt. Aus den besten 6 Teilnehmern werden dann zwei Teams geformt, welche die ETH am Regional Contest vertreten werden.

2. **Regional Contest:** Alle Teams eines Regional-Bereiches (für uns: Südwesteuropa) treten gegeneinander an. Ab jetzt wird in Teams programmiert, also 3 Leute und nur ein Computer. Der diesjährige SWERC (Southwestern European Regional Programming Contest) findet am 20./21. November in Madrid statt (selbstverständlich werden alle Reisekosten etc. übernommen). Die zwei bestplatzierten Teams unserer Regionals qualifizieren sich für das Weltfinale.
3. **World Finals:** Am Weltfinale des ACM ICPC treffen sich die besten Teams der verschiedenen Regionen und machen untereinander den World Champion aus. Dieses Jahr finden die World Finals in Ägypten vom 27.2.2011 bis 4.3.2011 statt.

Für den ACM ICPC sind Analyse-, Algorithmik- und Implementions-Skills gefragt. Desweiteren ist Teamfähigkeit enorm wichtig, denn jedem Team steht nur ein Computer zur Verfügung.

Der VIS-ACM führt im kommenden Herbstsemester wieder verschiedene Trainingscontests durch. Der erste davon findet bereits am Samstag den 25. September statt. Diese Trainingscontests sind insbesondere auch als Einführung gedacht und stehen allen offen, die interessiert sind. Um die beiden ETH Teams danach optimal auf den Regional Contest vorzubereiten, organisieren wir, zusammen mit dem Departement Informatik, eine Trainingswoche mit ehemaligen ACM World Champions aus Russland.

Ebenso kann dieses Herbstsemester wieder das ACM Lab belegt werden, das sich dem Einüben effizienter Programmiermethoden und Algorithmen anhand von Programmieraufgaben aus den vergangenen ACM Wettbewerben widmet.

Alle Informationen rund um das Training und den Wettbewerb an der ETH sind verfügbar unter [1]. Abschliessend ist noch zu sagen, dass wirklich alle Studenten, welche die Bedingungen erfüllen, angesprochen sind. Schon mehrmals haben sich Studenten aus dem ersten Semester (“Erstis”) oder aus anderen Studiengängen für den Regional Contest qualifiziert.

Andere Programmierwettbewerbe

Die Community rund um Programmierwettbewerbe wächst stetig an und es gibt neben dem ACM ICPC noch viele andere Wettbewerbe, an denen man sich messen kann.

Zwei Beispiele sollen hier etwas genauer untersucht werden, die Internationale Informatikolympiade und die Topcoder Algorithm Competitions.

Internationale Informatikolympiade

Die Internationale Informatikolympiade (IOI) kann als ACM-ICPC-Äquivalent auf gymnasialer Stufe gesehen werden. Aus über 80 Ländern kommen die jeweils vier besten Schüler zusammen um sich an anspruchsvollen Aufgaben zu messen. Im Gegensatz zu den ACM-Wettbewerben handelt es sich nicht um einen Team-Wettbewerb und als Sprachen sind C, C++ und Pascal erlaubt.

Die Vertreter der einzelnen Länder werden in den meisten Fällen in nationalen Wettbewerben ➔

ausgewählt, im Falle der Schweiz beispielsweise durch die SOI, die Schweizer Informatikolympiade[2].

Zu bemerken ist, dass das Niveau an der Internationalen Informatikolympiade, auch von einem universitären Standpunkt her gesehen, extrem hoch ist. Der Leser kann sich gerne mithilfe der diesjährigen Olympiadenaufgaben[3] selber ein Bild darüber machen

Topcoder

Wer schon zu lange studiert und so weder an der Informatikolympiade noch an dem ACM ICPC teilnehmen kann, für den gibt es noch Topcoder.

In regelmässigen Abständen werden Wettbewerbe angeboten, bei denen teilweise auch Geld gewonnen werden kann. Früher gab es Geldpreise für jeden Wettbewerb und einzelne Teilnehmer haben im Laufe der Zeit gute fünfstellige Gewinne eingefahren. In der letzten Zeit haben jedoch nur noch wenige Wettbewerbe Geldpreise. Erlaubte Sprachen sind C++, Java, C#, Visual Basic und Python. Die Wettbewerbe sind erheblich kürzer als an den ICPC Wettbewerben und dauern in der Regel etwa 90min. Dafür wird umso mehr Wert auf die Zeit gelegt, die man braucht um ein Problem zu lösen: je später man eine Lösung einsendet, desto weniger Punkte gibt es. Ein spezieller Aspekt von Topcoder sind die sogenannten Challenge-Phasen. Nach dem eigentlichen Programmieren können die Quellcodes der Kontrahenten betrachtet werden und, falls man meint man habe einen Fehler gefunden, herausgefordert werden, indem man ihm eine Eingabe serviert, die er nicht korrekt bearbeiten wird. Erfolgreiche Challenges werden mit Zusatzpunkten belohnt, erfolglose mit Maluspunkten bestraft.

Trainingsmöglichkeiten

Neben den Contests an sich gibt es auch noch haufenweise Trainingsmöglichkeiten. Die meisten basieren auf einem Online Judge, das heisst das Problem kann wie am richtigen Contest gelöst, implementiert und eingesendet werden. Der Judge sagt einem dann, ob die eingesandte Lösung korrekt war.

Ein paar Seiten im Überblick:

SPOJ (Sphere Online Judge[5]) bietet ICPC-ähnliche Probleme verschiedener Schwierigkeitsstufen. Es sind sehr viele Programmiersprachen erlaubt und für jedes gelöste Problem gibt es Punkte (der Schweizer Teilnehmer mit den aktuell meisten Punkten ist ein bekanntes VIS-Mitglied).

Uva (Universidad de Valladolid Online Judge[6]) hat eine annährend komplette Sammlung von ehemaligen ACM-ICPC-Wettbewerbsaufgaben. Der Judge akzeptiert Lösungen in C, C++, Java und Pascal.

Project-Euler[7] ist eine Reihe von mathematischen Problemen, die mit Hilfe von Computerprogrammen gelöst werden können und ist daher eher für die mathematisch-orientierteren Teilnehmer ansprechend. ◆

Terminübersicht

Für aktuelle Informationen, schreib dich auf der ETH-ACM-Mailingliste ein[8] und besuche [1]. Der provisorische Terminplan sieht wie folgt aus:

25.9.	Trainingscontest
2.10.	Trainingscontest
9.10.	**ETH Local Contest (mit anschliessendem Apéro von unserem Sponsor Quatico[9])**
18.–23.10.	Trainingswoche
20.–21.9.	Regional Contest in Madrid

Links

[1] http://www.icpc.ethz.ch/

[2] http://www.soi.ch/

[3] http://www.ioi2010.org/CompetitionTask.shtml

[4] http://forums.topcoder.com/

[5] http://www.spoj.pl/

[6] http://uva.onlinejudge.org

[7] http://www.projecteuler.net/

[8] acm-interessenten@lists.vis.ethz.ch

[9] http://www.quatico.com/

[i][A][E][T][H]

Informatik-Alumni ETH Zürich

Portait Letter 012
Stefan Arn

Text & Gestaltung: Felix Würsten

Wenn man im Frühjahr 2010 den Auftrag erhält, einen Informatikingenieur zu porträtieren, der in leitender Stellung bei der UBS tätig ist, erwartet man nicht unbedingt jemanden, der mit grosser Euphorie von seiner gegenwärtigen Tätigkeit erzählt – schliesslich steht die Grossbank immer noch im medialen Fokus und muss zurzeit die Altlasten der letzten Jahre bewältigen. Doch wenn man Stefan Arn gegenübersitzt, ist die Krise vor allem Chance. Der CIO der Geschäftseinheit Wealth Management & Swiss Bank IT sprüht regelrecht vor Energie und berichtet begeistert über die enorme Herausforderung, gemeinsam den «Supertanker» UBS, ein Unternehmen mit immerhin knapp 70 000 Mitarbeitern weltweit, wieder auf Kurs zu bringen. «Wenn man einen solchen Turnaround aktiv miterlebt, dann prägt einen das für das ganze Leben», ist Stefan Arn überzeugt. «In solchen Phasen zeigt sich, wo die Stärken eines Unternehmens wirklich liegen.»

Zur Informatik kam Stefan Arn vor allem aus zwei Gründen: Informatik ist eine junge Disziplin, und man kann bereits während des Studiums mit dem erworbenen Wissen Geld verdienen. «Als ich mich nach meiner Lehre als Chemielaborant entschloss, ein Studium zu absolvieren, kamen für mich verschiedene Fächer in Frage: Germanistik, Geschichte, Physik und Biochemie waren Gebiete, die mich auch interessiert hätten.» Den Ausschlag gab schliesslich eine Vorlesung von Konrad Zuse, an die sich Stefan Arn noch heute erinnert. «Dass ich einen Gründervater dieser Disziplin persönlich erleben durfte, hat mich beeindruckt.» Der Pioniercharakter der Informatik sei im praktischen Berufsalltag immer noch spürbar. «Wenn man schaut, wie Bauingenieure und Informatiker ihre Projekte abwickeln, dann liegen da Welten dazwischen.» Wenn Stefan Arn heute, mehr als 20 Jahre nach seinem Abschluss, an seine Studienzeit zurückdenkt, erinnert er sich vor allem an die Vorlesungen des Informatikpioniers Niklaus Wirth. «Er war mit seinem Denken der Zeit weit voraus, und vieles von dem, was er uns in seinen chaotischen Vorlesungen erzählte, habe ich erst im Nachhinein wirklich begriffen», erzählt Stefan Arn. «Noch heute profitiere ich bei der Umsetzung von Projekten von seinen Ideen.»

Dass Stefan Arn ein Mensch ist, der gerne unternehmerisch tätig ist und schwierige Situationen als besonders stimulierend empfindet, bewies er bereits früh in seiner Laufbahn. Noch während des Studiums gründete er 1988 das Unternehmen AdNovum Informatik. «Wir waren zunächst vor allem in der Werbe- und Grafikbranche tätig.» Doch schon bald geriet der Aufbau der Firma ins Stocken, durchlebten diese Branchen Anfang der Neunzigerjahre doch eine schwere Krise. «In kurzer Zeit verloren wir praktisch alle Kunden», erinnert sich Stefan Arn. Für das junge Unternehmen war dies eine bedrohliche Situation, da es damals bereits über 20 Mitarbeiter beschäftigte. Stefan Arn entschloss sich daher, andere Geschäftsfelder zu bearbeiten. AdNovum begann, Softwarelösungen im High-End-Bereich, z.B. Bankingapplikationen, Middleware- und Securitylösungen, zu entwickeln und konnte sich in diesem Gebiet erfolgreich etablieren. Zu den Kunden der Firma gehören neben Banken, Finanzdienstleistern und Versicherern beispielsweise auch Behörden wie das kantonale Steueramt Zürich oder das EJPD,

die sensible Daten schützen müssen. Der wirtschaftliche Erfolg gab ihm recht: AdNovum erlebte einen erfreulichen Aufschwung und wuchs auf 200 Mitarbeiter an. Neben dem Standort Zürich begann die Firma auch in Budapest, Bern und Kalifornien Geschäftseinheiten aufzubauen. Anerkennung für seine Leistungen erhielt Stefan Arn auch, als er 2003 als Ernst&Young Swiss Entrepreneur of the Year ausgezeichnet und im Jahr darauf als Schweizer Kandidat für die Auszeichnung als World Entrepreneur of the Year 2004 nominiert wurde.

Anfang 2007 entschloss sich Stefan Arn, seine Firma zu verkaufen und ein Jobangebot der UBS anzunehmen. «Ich spürte, dass die Zeit für einen Wechsel gekommen ist. Ab einem gewissen Punkt wird man als Eigentümer und Chef in Personalunion zum Problem des Unternehmens», hält er nüchtern fest. «Es fällt immer schwerer, die eigenen Vorstellungen zu hinterfragen.» Dass AdNovum Informatik immer noch erfolgreich im Markt tätig ist, ist seiner Ansicht nach die wahre Bestätigung, dass er als Unternehmer erfolgreich war. «Als Firmenchef muss es doch mein Ziel sein, das Unternehmen auf eine langfristig solide Basis zu stellen, so dass es auch ohne mich überleben kann», erläutert er seine Philosophie. «Wenn das nicht gelingt, hat man etwas grundlegend falsch gemacht.»

Bei der UBS ist Stefan Arn nun für einen vitalen Bereich der Firma zuständig, muss er doch zusammen mit seinem Team dafür sorgen, dass der Bereich Vermögensverwaltung mit entsprechender Software ausgerüstet wird. «Wenn die Vermögensverwalter ein neues Produkt lancieren wollen, brauchen sie entsprechende Softwareinstrumente, um beispielsweise die Risiken der Produkte zuverlässig einschätzen zu können», erläutert Stefan Arn. Gerade in der jetzigen Krisensituation seien die Anforderungen an die Bank massiv gestiegen. «Time to market wird aus IT-Sicht immer mehr zum alles entscheidenden Erfolgsfaktor», stellt Stefan Arn fest. «Nur wer innovative Angebote punktgenau und schneller als die anderen auf den Markt bringt, kann sich auf dem hart umkämpften Markt längerfristig behaupten.»

Bei aller Begeisterung für die Herausforderungen, die Stefan Arn zurzeit bei der Grossbank bewältigen muss, hat die Krise für ihn doch auch ihre Schattenseiten. So musste er aufgrund der hohen beruflichen Belastung das Präsidium des Branchendachverbandes ICTswitzerland an Rudolf Noser abtreten. Der Wechsel habe auch seine guten Seiten, räumt Stefan Arn ein, denn im Gegensatz zum Politiker Noser fehle ihm häufig die Geduld, die man bei politischen Entscheidungsprozessen und bei der Lobbyingarbeit brauche. Schwerwiegender wiegt für ihn hingegen, dass er für sein Hobby kaum mehr Zeit hat. In seiner Freizeit ist er nämlich ein begeisterter Rennfahrer. «Am liebsten fahre ich mit einem Tourenwagen Rundstreckenrennen in Monza oder Hockenheim», erzählt er. «In dieser Klasse kann man mit vernünftigem finanziellem Aufwand auf recht hohem Niveau Sport betreiben.» Und wer erlebt hat, wie Stefan Arn über die Chancen des Turnarounds bei der UBS spricht, ist nicht mehr wirklich überrascht, dass ein Rennen zu fahren für ihn pure Erholung bedeutet. «Wenn ich im Rennauto sitze, kann ich wunderbar abschalten, das ist für mich wie Meditation», meint er lachend. «Und wenn der Kopf leer ist, kommen mir plötzlich ganz unerwartet die besten Ideen.»

Biographie

Stefan Arn, Jahrgang 1961, schloss sein Studium an der ETH Zürich 1989 ab. Bereits im Jahr zuvor gründete er die Firma AdNovum Informatik AG, die im Bereich Entwicklung und Integration von Geschäftsanwendungen und Sicherheitssoftware tätig ist. Im Jahr 2007 verkaufte Stefan Arn die Firma und wechselte als Head of Application Development for Clients and Products zur Grossbank UBS. Seit Oktober 2009 ist er Chief Information Officer der Geschäftseinheit Wealth Management & Swiss Bank IT. Neben seiner beruflichen Tätigkeit ist er in zahlreichen Gremien engagiert, so etwa auch im Dachverband der IT-Branche ICTswitzerland, den er von 2006 bis 2009 präsidierte. Kontakt: stefan.arn@ubs.com

Update from Hochschulpolitik (University Politics)

CORINA BASSI — IT IS DEBATED WHETHER SHE STILL WORKS TOO MUCH

Students are directly affected by the decisions that are made in university politics. However, they often miss the opportunity to make their own opinions known, because they are not aware how much influence they could have.

Lecture evaluations

The lecture evaluations provide students with a chance to give feedback on their lectures. However, the questions mostly aren't very meaningful and far too general. There is the possibility to write comments, but how well they are discussed and considered depends on the professors.

The students should have the opportunity to give meaningful feedback on their lectures that must be considered by the professors. We are working on several ideas to ensure this: For the first two years of studies the concept of "Semestersprecher" has been introduced. Three students from each year are elected to act as intermediaries for students and professors with the task of organizing a discussion during the lecture without the professor being present after which they discuss the feedback with the professor. We are still working on a solution for lectures with two parts and different professors.

Additionally we are trying to add more meaningful questions to the written evaluation forms, and would also like to make them available to the students afterwards.

If you have an idea how we could improve this system let me know[1].

HoPo-Team

You have probably noticed that I talk about "us" all the time. The VIS has a team of students which discusses and handles topics concerning the computer science department. Among other things we represent the students opinion towards the department, try to improve the way you can give feedback about a lecture and talk to professors if students aren't happy with a lecture or exam.

Just send me an e-mail[1] if you want to join our team or if there is anything at the computer science department that needs our attention.

News from the department conference

- New head of the computer science department: Prof. Friedemann Mattern
- The internship is no longer part of the bachelor, but can be stated in your master report. It is no longer mandatory, but recommended.

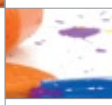
Contact

[1] hopo@vis.ethz.ch

Zukunftsforscherin?

Wenn sich deine Gedanken nicht nur mit dem Heute beschäftigen, sondern auch in die Zukunft wandern können, dann laden wir dich ein an unseren Innovationsprozessen für die Welt der Zukunft teilzunehmen. Der Tätigkeitsbereich der SCS ist die Computertechnologie. Hier sind wir stark und verändern dank innovativem Querdenken festgefahrene Strukturen, loten das Spektrum der Möglichkeiten aus und mischen Innovation und Technologie zu neuen marktfähigen Produkten.

Egal wie jung oder alt du bist, wenn du Innovation als Herausforderung und Leidenschaft definierst, dann bieten wir dir bei uns im Team tolle Einstiegsmöglichkeiten. Willkommen in der Welt des innovativen Querdenkens und professionellen Umsetzens.



Supercomputing Systems AG • Technoparkstrasse 1 • 8005 Zürich
brigit.richei@scs.ch • Telefon 043 456 16 60 • www.scs.ch

Perl Golf Step by Step

CHRISTIAN HELBLING AND REMO GISI — CODING FOR FUN

Removed code is debugged code. No one can argue about that. It is usually a good idea to remove code from programs by making them simpler and combining redundant parts into single, intuitive functions. It's part of the KISS[1] principle and it is also the philosophy of an interesting software website called suckless.org[2]. But what happens when we try to remove every single character of the program text? Well, it gets messy, unreadable and cryptic. But it's a lot of fun!

Warning!
This article contains code that is not meant to be used in any productive environment. Although most of the code is quite short it lacks most maintainability precautions like meaningful variable names, simple and understandable control flow, comments, and even line brakes, spaces, indentation and brackets. No robustness guarantees can be given. Executing the sample code could result in unexpected and possibly malicious behavior!

Perl is particularly well suited for making programs very small, which may be the reason why Perl Golf competitions[3][4] were invented on the newsgroup *comp.lang.perl.misc*[5]. The origin of the name is obvious: as in golf, the goal is to finish with the fewest (key-)strokes, and it is really amazing how small the programs can get. Just look at some solutions of past Perl Golf competition in the Perl Golf History book[6].

As an introduction to golfing in perl, we are going to take a look at a problem from the SPOJ subcontest called *SHORTEN*[7]. This contest has some small differences to pure perl golfing competitions as it also accepts other languages and has no special treatment of perl command line arguments, whereas most competitions allow writing things like "`-pa yourcode`" as a shortcut for "`#!/usr/bin/perl -pa`" followed by "`yourcode`" on the next line. The task, or "hole" in Perl Golf terms, we will look at is called *Pizza*[8] (see Box). If you don't understand some special perl variable, operator or type of syntax

Pizza — Problem statement

Abotrika is having a party because his team won the African Cup, so he is inviting his friends to eat some pizza. Unfortunately, Abotrika's friends can't eat an entire pizza, but all of them know exactly how much pizza they can eat, and insist on getting that exact amount. All of them want their amount of pizza in one slice. Also, Abotrika eats one whole pizza himself.
Their requests break down to three different pizza slices—either one quarter, or a half, or three quarters of pizza. Write a program that will help Abotrika to find out what is the minimal number of pizzas he has to order so that everyone gets the exact amount of pizza they want.

Input
First line contains an integer N ($0 \le N \le 10{,}000$), number of friends. In each of the next N lines there is the amount of pizza that each of Abotrika's friends wants to eat, that is the fraction $\frac{1}{4}$, $\frac{1}{2}$, or $\frac{3}{4}$.

Output
In the first and only line you should write the minimal number of pizzas Abotrika has order. Don't forget to order one complete pizza for Abotrika.

Examples

Input:
```
3
1/2
3/4
3/4
```

Output:
```
4
```

Input:
```
5
1/2
3/4
1/2
1/4
1/4
```

Output:
```
4
```

mentioned below we recommend looking into the perl documentation (which is pretty good by the way).

Listing 1 is a straightforward implementation for solving our pizza problem. The basic idea is that for all friends wanting to eat three-quarter slices we have to order a pizza anyway. From all these pizzas a quarter slice remains left over. Give those to our quarter-slice-wanting friends. Then cut as many pizzas in half as you can use for the half-pizza-wanting friends. That may not work out so one half could be left over which can be divided again into two quarter slices. After that we figure out how many pizzas are still needed for the remaining friends wanting quarter slices. Finally we add the pizza for Abotrika. ➔


VIS-Jassturnier
am 29. Oktober 2010
CABinett (StuZ² oben)
Türöffnung 18³⁰, Turnierbeginn 19⁰⁰
10.– / Person
Anmeldung in 2er-Teams
www.vis.ethz.ch/Jassturnier
VSETH
VERBAND DER STUDIERENDEN AN DER ETH
[i][a][e][t][h]
Informatik-Alumni ETH Zürich

Listing 1

```
$n=<STDIN>;                 # read number of lines
$halfs=0;
$quarters=0;
$three_quarters=0;
for($i=0;$i<$n;$i++) {      # for n lines:
    $_ = <STDIN>;           # read line into standard variable,
    if (/1\/2/) {           # means something similar to: if ($_ == "1/2")
        $halfs++;           # count halfs,
    } elsif (/1\/4/) {
        $quarters++;        # one quarter slices and
    } else {
        $three_quarters++; # three quarter slices (only these three values are possible)
    }
}

# at least $three_quarters pizzas are needed:
$pizzas = $three_quarters;
# use the rest of these pizzas with quarter slices:
$quarters -= $three_quarters;

# see how many pizzas can be cut in half:
$pizzas += ($halfs - ($halfs % 2)) / 2;

# and get one pizza for the final half if necessary:
if($halfs % 2) {
    $pizzas++;
    $quarters -= 2;         # use the rest of the half pizza for quarter slices
}

# if any quarter slices are left: see how many pizzas they need:
if($quarters > 0) {
    $pizzas += ($quarters - ($quarters % 4)) / 4;
    if($quarters % 4 > 0) {
        $pizzas++;
    }
}

# print the number of pizzas (including the one for Abotrika):
print $pizzas + 1;
```

Now that was over one kilobyte of code. Let's start removing unnecessary things. First of all we get rid of all the comments, unnecessary initializations (in perl uninitialized variables always behave like they were zero when used in an arithmetic operation) and most needless whitespace (but we still leave some of it for clarity). We switch to only using one-character variable names. In our case we just take the first character of the old variables. `<STDIN>` can be abbreviated to `<>`. We don't really need to know N as we assume the input files contain no additional lines which is true on the judge. So the first line in Listing 2 is just to skip the first line of the input. Then we use "`map`" to loop over all further lines (which are then stored in the standard variable `$_` inside this loop). There we use the return value of the matching regexes— ➔

esf2010

Juice

30.9.2010

Science City
ETH Hönggerberg

Pocketmaster | Manolo Panic
Not the Sensational Alex Harvey Band

Erstsemestrigenfest 2010

ETH Erstsemestrige 0.- | ETH-Legi 10.- | Legi 15.- | ohne Legi 20.-
20h - 3h | GRATIS Shuttlebus bis 4h | 5 Floors | Live Konzerte

www.esf.ethz.ch

which is one if it matches and zero otherwise—directly. Also we save us one variable and use `$p` (for `$pizzas`) straight away. And as a last trick we use a special notation for `if` blocks with just one statement. There we write the `if` part after the statement (see line 9 and 12 in Listing 2). To combine several things into one statement, the comma operator comes in handy.

Listing 2

```
<>;
map {
    $h+=/1\/2/;
    $q+=/1\/4/;
    $p+=/3\/4/;
}<>;
$q-=$p;
$p+=($h-($h%2))/2;
$p++,$q-=2if$h%2;
if($q>0) {
    $p+=($q-($q%4))/4;
    $p++if$q%4>0;
}
print$p+1;
```

We are down to about 170 characters. That's not bad for a start, but for Perl Golf this is still much too verbose. First of all we can save us the curly brackets of the first loop by using the postfix `for` notation (similar to the postfix `if` notation we applied before). Inside this loop we can also save some bytes by using a regex matching the slash from every input line and then using the special variables for pre- and postmatch (`$`` and `$'`) combined with the `?:` operator. After the loop we subtract everything we need from `$q` in one go and set it to zero if it got negative (that would be if Abrotrika had to throw away some of the quarter-slices). We don't use any explicit variable for the number of pizzas anymore but instead put the whole expression for that after the print statement. And in place of divisions by two and by four we use bit shift as that rounds down the result to the next integer at the same time. Finally the last semicolon is not needed.

Listing 3

```
<>;
/\//,$`-1?$t++:$'<3?$h++:$q++ for<>;
$q-=$t+$h%2*2;
$q=0 if $q<0;
print $t+($h+1>>1)+($q+3>>2)+1
```

We remove all our remaining whitespace and use the special variable `$-` which magically cannot have values below zero. This brings the 101 bytes of Listing 3 down to 81 in Listing 4.

Listing 4

```
<>;/\//,$`-1?$t++:$'<3?$h++:$-++for<>
;$--=$t+$h%2*2;print$t+($h+1>>1)+($-+
3>>2)+1
```

If you think this is about as far as we can get, you are wrong. With some new ideas we can beat this solution by quite a lot. If we count everything in quarters we can get much simpler code. Note that the half- and quarter-slices can be freely combined. If we have at least as many orders for quarter-slices as for three-quarter-slices we can just look at the total needed amount of pizza and round that up to the next integer. Otherwise `$t-$q` quarter-slices are wasted.

Now all we need is the total number of quarters and the difference of `$t` and `$q`. In the input loop we first evaluate the standard variable with `eval $_` for getting the whole fraction and not just the numerator we would get if we used `$_` directly. We compare the resulting value to one half using the spaceship operator `<=>`. This ➔

operator returns minus one if the left argument is smaller, plus one if the right argument is smaller and zero if both arguments are equal. In our code `$k` is therefore -1 for lines with "1/4", 0 for lines with "1/2" and +1 for lines with "3/4". The amount of quarters is exactly `$k+2` which we add up in the variable `$x`. And we directly get `$t-$q` in the variable `$y` when we add up the `$k`. If `$y` is positive this is the amount of wasted quarter-slices. Otherwise we set it to zero using again an assignment to `$-`. To the total number of needed quarters `$x` we add the wasted quarters `$-`, four quarters for Abotrika and three additional quarters to make our final shift operator behave like rounding up.

Listing 5

```
<>;$k=eval$_<=>.5,$y+=$k,$x+=$k+2for
<>;$-=$y;print$x+7+$->>2
```

After such a big change there is usually some new redundancy around. We see that `$x` and `$y` differ by just 2*N. Instead of using again a variable for N we can use another special variable `$.`. This variable holds the current line number of the input file. After the loop we have read N+1 lines so `$.` is equal to N+1. So we have `$x==$y+2*$.-2` which we plug in into our expression for the result. We now need the result of the spaceship operator only once so we can save us the variable `$k`. This gets us from the 60 characters from Listing 5 to 50 characters in Listing 6.

Listing 6

```
<>;$y+=eval$_<=>.5for<>;$-=$y;print$y
+$.*2+5+$->>2
```

Now we have a program that is almost as short as it can get. But we can still apply three more tricks and lose one character for each of them. The first trick is to swap the operands of the spaceship operator so that we can make the `eval` command use the standard variable implicitly as its argument. Of course this also negates the operator's result, so we use `$y-=` instead of `$y+=`. The second trick is to inline the assignment to `$-` using brackets. The last trick is a bit more advanced: we get rid of the code to skip the first line of the input (`<>;`). Let's look what changes if we just leave that out. If N ≥ 1 our variable `$y` gets one too big. We compensate that by subtracting one from our constant 5 and changing `$-=$y` into `$-=$y-1`. Now in

the sad case that Abotrika has no friends we have `$y==-1`, `$-==0` and `$.==1` which luckily gives us still the correct result of one pizza.

Listing 7

```
$y-=.5<=>eval for<>;print$y+$.*2+4+
($-=$y-1)>>2
```

So now we have solved this problem in 47 characters. This is pretty compact. At the time of writing the best submissions on SPOJ are 45 characters long. Maybe you can find the trick(s) to squash the other missing two characters or even more! And if you do: tell us :)

Happy perl golfing! ◆

Links

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

[2] http://suckless.org/manifest/

[3] http://www.perlmonks.org/index.pl?node_id=21442

[4] http://codegolf.com/

[5] http://groups.google.com/group/comp.lang.perl.misc

[6] http://terje2.frox25.no-ip.org/perlgolf_history_070109.pdf

[7] http://www.spoj.pl/SHORTEN/

[8] http://www.spoj.pl/SHORTEN/problems/PIZZAS/

Picture Credits

Cover photograph
© Elena Schweitzer | fotolia.com

Page 17: "The King's Game"
by Levente Fulop, CC-2.0-by
flickr.com/photos/levyfulop

Page 18: "Game Over"
by Mykl Roventine, CC-2.0-by
flickr.com/photos/myklroventine

Pages 28, 42: Warning sign
© Icons Land, Demo License
www.icons-land.com

Pages 28-31: Yellow star
by Sam Butcher III, royalflushxx.deviantart.com

Pages 28-31: Grey star
by Webdesigner Depot
www.webdesignerdepot.com

Page 37: "I program in my sleep"
by Rachel Johnson, CC-2.0-by-nd
flickr.com/photos/rachel-johnson

Page 51 background: "Star Gazing"
by Steve Jurvetson, CC-2.0-by
flickr.com/photos/jurvetson

Impressum

VISIONEN

Magazin des Vereins der Informatik Studierenden an der ETH Zürich (VIS)

Ausgabe September 2010

Periodizität	6x jährlich
Auflage	1400
Jahresabonnement	CHF 25.–

Chefredaktion
Fabian Hahn
visionen@vis.ethz.ch

Layout
Daniel Saner
layout@vis.ethz.ch

Inserate
Jérémie Miserez
inserate@vis.ethz.ch

und freie Mitarbeiterinnen und Mitarbeiter

Anschrift Redaktion & Verlag
Verein Informatik Studierender (VIS)
CAB E31
Universitätsstr. 6
ETH Zentrum
CH–8092 Zürich

Inserate (4-farbig)

½ Seite	CHF 850.–
1⁄1 Seite	CHF 1500.–
1⁄1 Seite, Umschlagsseite (U2)	CHF 2500.–
½ Seite, Rückumschlag (U4)	CHF 2500.–

Andere Formate auf Anfrage.

Druck
Binkert Druck AG
5080 Laufenburg
http://www.binkert.ch/

VIS

VSETH
VERBAND DER STUDIERENDEN AN DER ETH

Der VIS ist Teil des Verbandes der Studierenden an der ETH (VSETH).

Puzzled

BARBARA KELLER — HOFFT FÜR DIE GANS
ROGER WATTENHOFER — SETZT AUF DEN WOLF

Die Gans und Meister Isegrim

Es war einmal eine kleine süsse Gans, welche gerne auf ihrem kreisrunden See umherschwamm. Eines Tages kam ein sehr hungriger Wolf vorbeispaziert. Der dachte sich: "Hmmm... frische Gänseleber wär jetzt ein gar feines Abendbrot. Wie gut, dass ich an Land 4 mal so schnell rennen kann, wie die Gans schwimmen kann." Die Gans muss von Zeit zu Zeit an Land um etwas zu essen. Erreicht die Gans das Ufer ohne bereits vom Wolf erwartet zu werden, kann sie dem Wolf entfliegen.

Nun ist Deine Hilfe gefragt: Gibt es eine Taktik, welche die Gans anwenden kann, so dass sie dem hungrigen Wolf, welcher sich nicht in den See wagt, entwischen kann?

Sowohl die Gans als auch der Wolf möchten ausserdem gerne wissen, wievielmal schneller der Wolf höchstens sein darf, so dass ihm eine schlaue Gans gerade noch entwischen kann.

Kurz bevor der Wolf das Ufer erreicht, erinnert er sich, dass er Wasser zwar nicht besonders mag, dass er aber sehr wohl schwimmen kann. Allerdings ist er der Gans geschwindigkeitsmässig im Wasser nicht mehr überlegen, sondern schwimmt genau gleich schnell wie sie. Wenn nun um den Teich ein Zaun ist, die Gans also nicht davonfliegen kann, gäbe es dann trozdem eine Möglichkeit für die Gans, nicht als herrliches Abendessen des Wolfes zu enden?

Hilfestellungen für die Gans / den Wolf werden gerne hier entgegengenommen:
puzzled@vis.ethz.ch
Die besten eingereichten Lösungen werden mit einem kleinen Preis honoriert.
In den nächsten Visionen werden die Lösungen präsentiert.

AZB
PP/Journal
CH – 8092 Zürich

Falls unzustellbar, bitte zurück an:
Verein Informatik Studierender
CAB E31
Universitätsstr. 6
ETH Zentrum
CH-8092 Zürich