Zeitschrift: Visionen : Magazin des Vereins der Informatik Studierenden an der ETH Zürich

Herausgeber: Verein der Informatik Studierenden an der ETH Zürich

Band: - (1993)

Heft: 12-1

Heft

Nutzungsbedingungen

Die ETH-Bibliothek ist die Anbieterin der digitalisierten Zeitschriften auf E-Periodica. Sie besitzt keine Urheberrechte an den Zeitschriften und ist nicht verantwortlich für deren Inhalte. Die Rechte liegen in der Regel bei den Herausgebern beziehungsweise den externen Rechteinhabern. Das Veröffentlichen von Bildern in Print- und Online-Publikationen sowie auf Social Media-Kanälen oder Webseiten ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Mehr erfahren

Conditions d'utilisation

L'ETH Library est le fournisseur des revues numérisées. Elle ne détient aucun droit d'auteur sur les revues et n'est pas responsable de leur contenu. En règle générale, les droits sont détenus par les éditeurs ou les détenteurs de droits externes. La reproduction d'images dans des publications imprimées ou en ligne ainsi que sur des canaux de médias sociaux ou des sites web n'est autorisée qu'avec l'accord préalable des détenteurs des droits. En savoir plus

Terms of use

The ETH Library is the provider of the digitised journals. It does not own any copyrights to the journals and is not responsible for their content. The rights usually lie with the publishers or the external rights holders. Publishing images in print and online publications, as well as on social media channels or websites, is only permitted with the prior consent of the rights holders. Find out more

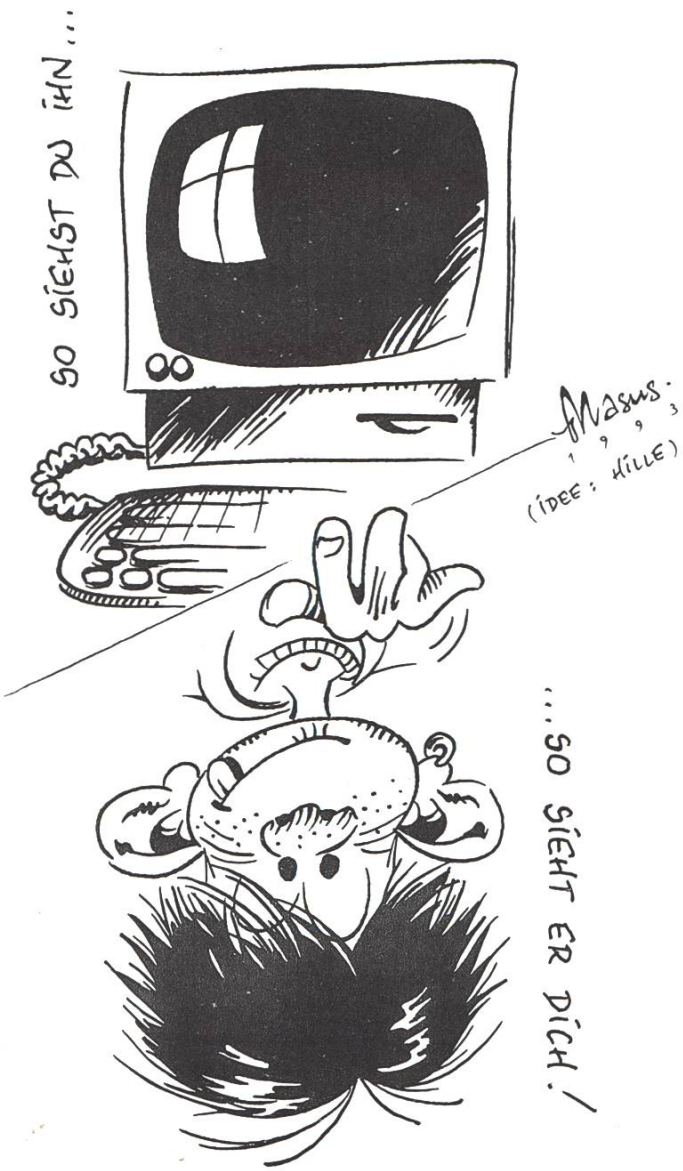
Download PDF: 20.08.2025

ETH-Bibliothek Zürich, E-Periodica, https://www.e-periodica.ch

Visionen

12/1

Dezember 93
Januar 94



Bieridee: Übung macht den Meister
Swansea: Fast gewonnen!
C-Kurs: Erster Teil!

Visionen Herausgegeben vom Verein der Informatikstudierenden an der ETH Zürich

Adressen

Aktuar: Stefan Rohmer
Keltenstrasse 6, 8044 Zürich
Tel. 01 / 251 34 51
e-mail: stefan@vis.inf.ethz.ch

Exkursionen: Boris Nordenström
Hardstrasse 324, 8005 Zürich
Tel. 01 / 273 24 80
e-mail: banorden@iiic.ethz.ch

Feste & Kultur: Frank Möhle
Dielsdorferstrasse 7, 8155 Niederhasli
Tel. 01 / 851 03 21
e-mail: fmoehle@iiic.ethz.ch

Präsidentin: Grete Danielsen
Dohlenweg 26, 8050 Zürich
Tel 01 / 302 48 97
e-mail: gcdaniel@iiic.ethz.ch

Quästor: Daniel Kluge
Irringersteig 3, 8006 Zürich
Tel. 01 / 252 04 14
e-mail: dankluge@iiic.ethz.ch

Redaktor: Patrick Leoni
Sandstr. 6, 8610 Uster
Tel. 01 / 940 05 14
e-mail: pleoni@iiic.ethz.ch

Verleger: Hans Domjan
Kapfhalde 3, 6020 Emmenbrücke
Tel. 041 / 53 68 83
e-mail: hdomjan@vis.inf.ethz.ch

Visinfo(Infosystem): Michel Müller
Rheinländerstr. 15, 4056 Basel
Tel. 061 / 321 81 23
e-mail: mimuelle@iiic.ethz.ch

Prüfungen und Unterricht:
Leonhard Jaschke
Südstrasse 67, 8008 Zürich
Tel. 01 / 383 60 55
e-mail: ljaschke@iiic.ethz.ch

Impressum

Herausgeber:
Verein der Informatikstudierenden an der ETH Zürich.

Verleger: Hans Domjan
Redaktor: Patrick Leoni

Adresse Verlag & Redaktion:
VIS
Verein der Informatikstudierenden
Haldeneggsteig 4, IFW B29
ETH Zentrum
8092 Zürich

Tel: 01 632 72 12 (Mo-Fr, 12.15-13.00)
Fax: 01 262 39 73
e-mail: vis@iiic.ethz.ch
Postkonto 80-32779-3
Präsenzzeit: Mo-Fr: 12.15-13.00

Auflage: 1400
Inseratepreise:

1 Seite	s/w	SFr. 500.–
1 Seite	Farbe	SFr. 750.–
1/2 Seite	s/w	SFr. 250.–
1/4 Seite	s/w	SFr. 150.–

Redaktions- und Anzeigeschluss für die nächste Ausgabe:
Freitag, 21. Januar 1994

Visionen

© 1992, 1993 by
Verein der Informatikstudierenden

Hei Folkens!

Eine Unmenge an Fanpost und Klagen seit den letzten Visionen hat mich dazu veranlasst, wieder das populäre "Hei Folkens" anstelle des "Hoi Zäme" an den Kopf dieser Seite zu setzen. Hoffentlich ist die Mehrheit jetzt zufrieden...

Inzwischen sind die letzten Käseschwaden vom FIGUGEGL aus dem StuZ und hoffentlich auch aus den Klamotten der Teilnehmer verschwunden. Das diesjährige Fondue war dank Franks bester Organisation (ein *tusen takk* an Frank & Company) so gelungen, dass ein Türsteher notwendig war, um überflüssige Partyfreaks in die unteren Räumlichkeiten des StuZ zu dirigieren.

Wer geglaubt hat, dies sei alles an der Partyfront für dieses Jahr, ist entweder ein Frischling (weibl. Form heuer nicht notwendig) oder hat die Alzheimersche Krankheit. Denn am 21. Dezember gibt's Rocky X-mas (ex Heavy X-mas) für alle, die noch nicht ihre Koffer für den Südseetrip gepackt haben. Der volle Name der Live-Band *BJ* kann bei Frank erfragt werden... :-))

Nun zu den ernsteren Dingen im Leben: die Prüfungsvorbereitungen für die Vor- und Schlussdiplome. Tiefenpsychologische Untersuchungen am *Institutt for Hjerneforskning* an der NTH haben gezeigt, dass das Phänomen der Massenhysterie mit grossem Erfolg für Lernzwecke eingesetzt werden kann. Wir wagen als erster VIS europaweit, dieses Verfahren in unseren Lerngruppen anzuwenden. Dafür benötigen wir Tutoren (Professoren, Assistenten, Studenten) mit praktischen und theoretischen Kenntnissen in den verschiedensten Prüfungsfächern, welche die Lernwilligen (Professoren, Assistenten, Studenten) mit Geduld und Ausdauer betreuen, sowie Erfolg und Misserfolg mit ihnen teilen. Anmeldungen von Tutoren und Lernwilligen werden im VIS-Büro angenommen. In dieser Ausgabe der Visionen könnt ihr einen Anmeldetalon und einen genaueren Text von unserem Prüfungsexperten Leo finden. Keine Hemmungen!

Wie Ihr wahrscheinlich alle nicht wisst, wird der VIS am 26. April 1994 zehn Jahre alt. Wir halten dies für einen exzellenten Grund, ein prächtiges Fest zu feiern. Unsere Vision ist, ein OK (*organisasjons kommite*) für dieses Jubiläum aufzustellen. Alle, die mithelfen und mitorganisieren wollen, gute Ideen und Vorschläge haben, Zeit und Drive mitbringen, sind herzlich eingeladen im OK mitzumachen. Auch hier heisst es "Ab zum VIS-Büro", wenn Du Dich angesprochen fühlst.

Am Montag, den 24. Januar 1994 findet mit der Unterstützung der Abteilung IIIC die alljährliche Kontaktparty statt. *Tusen takk* an das Organisationsgenie, unsere Abteilungssekretärin Frau Hilgarth. Falls Ihr Euch über die rosigen Berufsaussichten erkundigen wollt, eine Praktikumsstelle oder Eure erste Stelle sucht, oder einfach 'mal neugierig vorbeischauen wollt, ist die KP '94 der richtige Ort.

Fröhliche Weihnachten und es guets Neuis

GOD JUL OG GODT NYTT ÅR ønsker

Grete

Visionen — Redaktionsschlüsse 1994

Der VIS-Vorstand hat die Redaktionsschlüsse für das Jahr 1994 neu festgelegt. Dabei berücksichtigten wir (d. h. vor allem ich, Patrick... :-) im besonderen die folgenden Punkte:

- Das auf den Visionen aufgedruckte Datum sollte mit dem tatsächlichen Erscheinungsdatum etwas zu tun haben. Diese schlichte (und recht eigentlich selbstverständlich klingende) Feststellung ist in der Vergangenheit ein wenig in den Hintergrund getreten. Des öfteren erschien etwa die Januar-Ausgabe erst gegen Mitte Februar. Obwohl für uns die Erscheinungsdaten einigermassen klar definiert waren, korrelierten diese mit den angepeilten Zeiträumen mitunter nur schwach.
  Soweit die Antwort auf: "*He, Pleoni, was soll das – die Mai-Visionen Anfang Juni?*"

- Der mühsame, wiederholt beklagte und doch ziemlich unvermeidliche Erscheinungsdelay zwischen Redaktionsschluss und Erscheinen der Visionen soll einerseits berechenbarer, andererseits als Institution festgelegt werden. Die Zeit schwankte zwischen einer knappen und drei (3) guten Wochen. Diese Frist kann zwar prinzipiell nicht mit Sicherheit auf eine gewisse Zeit limitert werden (das letzte Mal streikte das Haus-Netzwerk und VPP, und dann wurden wir unverhofft und unvermittelt mit einer neuen, unbekannten IP-Adresse beglückt...), jedoch sollte sie immer auf weniger als zwei Wochen gedrückt werden können.
  Die hierzu passende, manchmal ostentativ wiederholte Frage lautet: "*He, Pleoni, wann kommen denn die neuen Visionen?*"

Die daraus abgeleitete Idee ist nachvollziehbar: *Jede Ausgabe der Visionen soll am Anfang des aufgedruckten Monats erscheinen.* Warum das uns nicht schon lange eingefallen ist? Fragt mich etwas Einfacheres... Eine sofortige Konsequenz hat dieser Entschluss allerdings. Wie der geschätzte Leser, die geneigte Leserin zweifelsfrei festgestellt hat, ist diese Ausgabe auf den Zeitraum Dezember '93-Januar '94 gemünzt. Der ursprüngliche Redaktionstermin für den Februar wird von Ende Februar auf Mitte Januar vorverlegt, erscheinen wird die Februar-Ausgabe dann Anfang Februar.
Die Liste mit den neuen Redaktionsschlüssen ist im VIS-Büro ausgehängt – zu finden ist sie ausserdem in dieser Ausgabe.

pal, Red.

Bewerben – aber richtig: Abgesagt

Die Veranstaltung "Bewerben – aber richtig" musste von den (externen) Organisatoren abgesagt werden. Wegen eines Krankheitsfalles des Hauptreferenten kann sie frühestens im nächsten Jahr wieder in Erwägung gezogen werden. Der VIS-Vorstand wird dann seine Mitglieder über die neu angesetzten Termine ins Bild setzen und ihnen die Möglichkeit geben, sich wieder auf eine Warteliste zu setzen.

Der VIS-Vorstand möchte hiermit seinem Bedauern über die Verhinderung Ausdruck verleihen sowie bekräftigen, wie wichtig solche Veranstaltungen für die Zeit nach dem Abschluss des Studiums für jedeN einzelneN StudentIn sind. Da beim ersten Ansetzen dieses Vortrags nicht mit dem grossen Interesse seitens der Studierenden gerechnet weren konnte, musste die Veranstaltung umgeplant werden, und es wurden separate Termine für die einzelnen Abteilungen festgelegt. Bevor aber die Informatiker an der Reihe waren, erkrankte der Referent.

Das Rektorat der ETH hat aber seine schriftlichen Unterlagen erhalten, sie vervielfältigt und uns überlassen. Wir haben diese Kopien nun erhalten und können sie an interessierte StudentInnen weitergeben.

Wir wünschen dem Referenten gute Besserung und hoffen auf ein baldiges Wiederansetzen der Veranstaltung.

Für den VIS-Vorstand: pal

Redaktionsschlüsse 1994

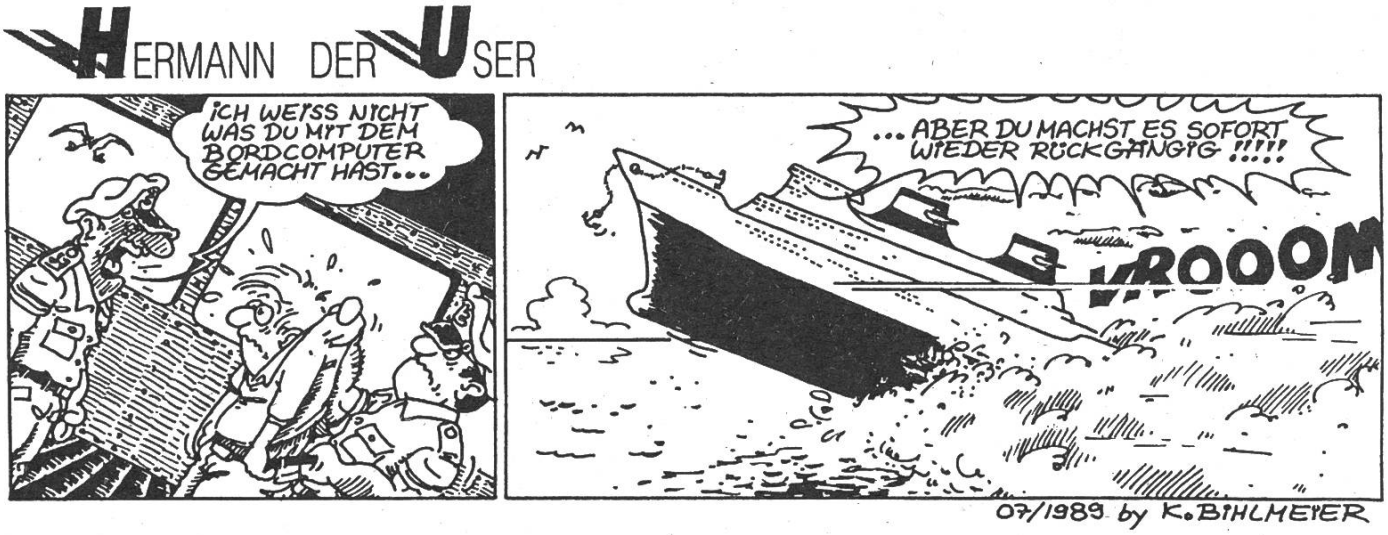
Ausgabe	*Redaktionsschluss*	*Erscheinen*
2 / 94	**21. Januar '94**	Anfang Februar
3-4 / 94	**18. Februar '94**	Sem'ende, Anfang März
5 / 94	**8. April '94**	Sem'beginn, Mitte April
6 / 94	**20. Mai '94**	Anfang Juni
7-8 / 94	**17. Juni '94**	Semesterende, Ende Juli
9-10 / 94	**19. August '94**	Semesterferienmitte
11 / 94	**21. Oktober '94**	Sem'beginn, Anfang Okt
12 / 94	**18. November '94**	Anfang Dezember
1/95	**16. Dezember '94**	Anfang Januar '95

NCR-Preis 1993

Otto Mayer und Hanspeter Waldegger haben für ihre Diplomarbeit "Föderierte Transaktionsverwaltung aufgesetzt auf relationalen Datenbanken" den ersten Preis der NCR-Stiftung erhalten. Herzliche Gratulation!

Die NCR-Stiftung prämiert Diplom- und Lizentiatsarbeiten, die sich mit der Anwendung von Informatik befassen. Die vorliegenden Arbeiten, durchgeführt am Institut für Informationssysteme bei Prof. Schek und betreut durch W. Schaad, untersuchen die Integrierbarkeit von bestehenden relationalen Datenbanken. Insbesondere wurde ein Konzept für die Unterstützung globaler Transaktionen bei gleichzeitiger Ausführung von autonomen, lokalen Transaktionen entwickelt. Das Ziel ist der Aufbau eines föderierten Datenbanksystems mit verschiedenartigen Komponentensystemen.

Mitgeteilt durch W. Schaad, Informationssysteme



1984 ... 1994

Unser Fachverein feiert Geburtstag
und plant eine

Megafete

(kein Aprilscherz: Am 26. April 1994)

Hast Du Lust, da mitzuplanen und
Schuld daran sein, dass diese Fete
zur

Gigafete

(auch kein Aprilscherz: immer noch am 26. April 1994)

wird?

Dann melde Dich doch bitte *jetzt*
beim VIS als OK-Mitglied.

10 Jahre

BBDO/BC

PowerBook.

Dein Begleiter fürs Studium.

COMPUTER-LADEN

Bucheggplatz/Rötelstrasse 135
8037 Zürich, Tel. 01/362 72 90

Autorisierter Fachhändler

The Knights who say "Ni!"
oder
Zehn Studenten in Swansea

Nur knappe sechs Tage nach unserem Sieg an der Ausscheidung in Zürich hob das Flugzeug in Kloten ab, um uns nach Grossbritannien zu bringen, wo wir an der europäischen Ausscheidung des ACM-Programmier-Wettbewerbs teilnehmen würden. Kaum gestartet stach das Flugzeug aus dem Nebel des Schweizer Mittellands hervor und trug uns bei strahlendem Sonnenschein nach London. Von dort ging es weiter mit der Bahn nach Swansea, einer mittelgrossen Stadt an der Süd-Küste von Wales. Im Zug lasen wir Monty Pythons. Lesen ist richtig; denn Frank hatte ein Skript von vier bekannten Filmen der Chaoten-Truppe. "The Holy Grail" fasziniert uns so, dass wir uns fortan mit "Ni!" begrüssten. Wir zehn Ritter waren die beiden Siegerteams aus dem ACM-Wettbewerb in Zürich. Dazu kam noch Frank und Hans von VIS, die alles organisiert hatten und ein paar Sponsoren ausgepresst hatten, um die Reise zu finanzieren.

Unsere Rüstung war eine Reisetasche mit Firmenlogo und ein T-Shirt, das vorne das VIS-Logo aufgedruckt hatte und hinten die Namen der grossen Sponsoren. Wir sahen aus wie wandelnde Plakatsäulen.

Die Landschaft in Südwales ist recht merkwürdig: Viele Ortschaften sind auf oder an einem Hügel erbaut und ziehen sich in Doppelreihen, links und rechts einer Strasse, den Hängen entlang. Es herrschen dunkelgrüne und rötliche Töne vor und man sieht weit und breit keinen einzigen Baum. Diese wurden schon vor längerer Zeit für den Bergbau abgeholzt.

In Swansea kam bald das Gefühl auf, dass die Organisation des Wettbewerbs nicht ganz perfekt ist. Im Hotel, wo wir untergebracht waren und wo auch ein Teil der Veranstaltung stattfinden sollte, wusste man weder von einer Transport-Möglichkeit etwas, die die Veranstalter organisiert haben sollten, noch tauchte um vier Uhr jemand auf für die offizielle Einschreibung der Teams. Die Dame an der Reception konnte sich immerhin schwach erinnern, den Begriff "ACM" schon gehört zu haben.

Wir erkannten auch, dass Hilton nicht gleich Hilton ist. Obwohl wir in einem solchen Hotel untergebracht waren, würden andere Hotels mit dem gleichen Namen wohl wünschen, nicht im gleichen Satz mit dem in Swansea erwähnt zu werden. Meist sah man erst auf den zweiten Blick, dass die Standards hier mit anderen Ellen gemessen werden. Am zweiten Tag erlebten wir dort, wie die Angestellten, bewaffnet mit Sprays, den üblen Geruch einer "Pizza" bekämpften, die man noch Stunden später neben der WC-Schüssel antreffen konnte. Und wenn man sich im Restaurant auf einen Stuhl setzte, überkam einen das schmuddelige Gefühl, das jedem aus der Zürcher S-Bahn bekannt ist.

So verliessen wir das Hotel bald wieder und suchten uns ein italienisches Restaurant in der Stadt, um ein feines Nachtessen zu uns zu nehmen. Danach wanderten wir etwas in der Stadt herum, sahen uns den Hafen an und stachen in ein Pub, um die ersten Biere zu testen. Es sollten weitere folgen.

Die neue Regel

Am Samstag fand das Seminar statt, das auch zum Wettbewerb gehört. ACM hatte geschrieben, dass man einen Teil der Reisekosten zurückerstattet bekomme, wenn man am Seminar teilnimmt. Es stellte sich jedoch heraus, dass ACM schon zu viel Geld ausgegeben hatte für die Seminar-Räume im Hilton-Hotel und darum nichts bezahlen konnte. Es bleibt unklar, wieso die britischen Veranstalter beschlossen hatten, das Seminar nicht an der Universität abzuhalten, wo sie die Räumlichkeiten bestimmt umsonst bekommen hätten.

Im Seminar wurden wir zuerst mit zwei Vorträgen gelangweilt. Interessanter war dann, was Ruth zu erzählen hatte, die als Vertreterin des ACM von Amerika gekommen war. Sie machte einigen Teams grosse Hoffnungen mit der Ankündigung einer neuen Regel, die besagte, dass von den zwei Teams, die nach Amerika an den Final reisen, höchsten eines pro Land dabei sein darf. In den letzten Jahren waren immer nur holländische Teams nach Amerika gefahren, was bei der total holländischen Uebermacht nicht erstaunt. Auch dieses Jahr stammten die Hälfte der Teams aus Holland. Unser Team begann, sich leise Hoffnungen zu machen, da das letzte Team der ETH, das an der europäischen Ausscheidung teilgenommen hatte, das beste nicht holländische Team wurde. Aber immerhin galt es ausser den anderen nicht holländischen Teams auch noch das zweite Team der ETH zu schlagen.

Dann wurden unsere bösen Gefühle, was die Organisation betrifft, weiter gefestigt. Der walisische Organisator, ein junger, rothaariger Typ mit Bart, wirkte inkompetent und überfordert, musste organisatorische Probleme eingestehen, wusste von der neuen Regel nichts und hatte seine ganze Autorität schon nach einer knappen Minute verspielt.

Gegen Abend wurden wir von einem Bus, der nicht gross genug war, um alle Teilnehmer aufzunehmen, an die Universität gefahren, wo die Computer standen, die wir am folgenden Tag während des Wettbewerbs brauchen durften. Hier wurden unsere schlechten Vorahnungen Tatsache. Nachdem sich das allgemeine Chaos etwas gelegt hatte und wenigstens die meisten Teams vor einem Rechner sassen, mussten wir feststellen, dass die Directories auf den Rechnern für alle lesbar waren, Netzwerk-Aktivitäten nicht überwacht oder unterbunden waren und der Systemmanager ebenfalls zur Species der überforderten Personen gehörte. Schon begannen die ersten, von anderen Teams ein Remote-Screendump zu machen und anzusehen. Zwar wussten die Organisatoren, dass es auf einer typischen Unix-Umgebung Netzwerk-Funktionen gibt, aber sie hatten rein gar nichts unternommen, um einen fairen Wettbewerb garantieren zu können. Sie erwähnten nur immer, dass wir froh sein könnten, dass die Computer überhaupt funktionierten und dass sie viel Mühe gehabt hätten, sie zum Laufen zu bringen. Das Letztere wunderte uns nicht.

Auf dem Rückweg liessen wir uns in der Stadt absetzen. Für das Nachtessen suchten wir uns ein Steak-House aus. Für das oder besser die Biere suchten wir wieder ein gemütliches Pub auf, wo wir nach elf Uhr hinausgeworfen wurden, nicht weil wir uns schlecht benommen hatten, sondern weil ein britisches Pub eben um Elf schliesst. Das war wahrscheinlich auch besser so. Denn der nächste Tag war ja der grosse Tag für uns und ein bisschen Schlaf konnte nicht schaden.

Der grosse Tag

Am Sonntag ging es um die Reise nach Amerika, die der grosse Preis der Ausscheidung war. Eigentlich war schon die Reise nach Swansea unser Preis und wir hätten uns zurücklehnen und den Preis geniessen können. Das liess aber unser Ehrgeiz nicht zu.

Der Wettbewerb hatte etwa die gleichen Regeln wie in Zürich. Es wurden mehrere Aufgaben ausgeteilt, die es in der gegebenen Zeit mit drei Personen und einem Computer zu lösen galt. Als Programmiersprache waren Pascal und C erlaubt. Wer am meisten Aufgaben löste, sollte gewinnen. Wenn mehrere Teams gleich viele Aufgaben lösen, so entschied die Zeit, wobei die Abgabe eines Programms, das noch nicht funktionierte, ein paar Strafminuten kostete.

Es fuhren also alle vom Hotel wieder an die Universität und kämpften im allgemeinen Chaos um einen Rechner. Mit viel Verspätung wurden die Aufgaben verteilt und der Wettbewerb begann. Recht entspannt machten wir uns an die Arbeit und froren vor uns hin, weil der Wind ziemlich ungehindert durch das Gebäude hindurch blies. Während unsere Köpfe langsam warm wurden und zu rauchen begannen, hiess es für die Coaches der verschiedenen Teams warten, warten, essen, warten und nochmals warten. Sie konnten sich die Aufgaben ebenfalls ansehen. Einer der Coaches, ein holländischer Professor(!), erklärte allen Interessierten, die Lösung einer der kniffligeren Aufgaben. Dagegen gibt es an sich nichts einzuwenden. Doch die Teilnehmer konnten sich ganz legal und ungehindert mit ihren Coaches treffen. Alle verpflegten sich ja am gleichen Ort. Und es gab auch kein Merkmal, woran man einen Coach als solchen hätte erkennen können.

Wir erlebten ähnliche Dinge wieder, die wir schon in Zürich erlebt hatten. Man gibt eine Aufgabe ab und bekommt die Meldung, dass sie nicht korrekt gelöst sei. Dann denkt man das Programm noch einmal durch, ändert dies und das und hat doch nicht das Gefühl, einen Fehler gefunden zu haben, bis dann die Meldung kommt, dass die Aufgabe bei der ersten Abgabe doch richtig gewesen war. Weiter gibt es Aufgaben, die wollen einfach nicht gelingen. Man hat sich alles gut überlegt, sieht keinen Fehler mehr und doch funktioniert es ganz offensichtlich nicht.

Am Schluss kommt die unvermeidliche Frustration. Man hat noch ein Programm, das nicht funktionieren will, und ein zweites, das schon fast fertig auf dem Papier steht. Aber die Zeit ist um. Man denkt, hätte man doch dies oder jenes getan, und wenn dies gewesen wäre, dann hätte man vielleicht, und vergisst darüber fast, dass man ja einiges geleistet hat, nämlich mehrere Aufgaben richtig gelöst hat.

So assen wir im Treppenhaus, dem offiziellen Verpflegungsort, wo es ziemlich penetrant nach Pissoir roch, noch ein paar Brötchen und liessen unsere Köpfe wieder ein wenig abkühlen. Danach gingen wir in den Saal, wo die Rangverkündigung stattfinden sollte.

Die Rangliste

Nach etwa einer halben Stunde tauchte der bärtige Chaot auf und verkündete, dass es bis zur Rangverkündigung noch eine Weile dauern werde. Offenbar gab es einige Streitfälle um Lösungen, von

denen gewisse Teams glaubten, sie seien richtig.

Nur eines wusste er schon mit Sicherheit, dass nämlich die Regel, dass höchstens ein Team pro Land nach Amerika darf, in diesem Jahr an der europäischen Ausscheidung noch nicht gilt. Denn er hatte von ACM keine offiziellen Unterlagen über die neue Regel erhalten. Und darum galt sie also nicht. Damit schwanden auch unsere kühnen Träume, nach Amerika gehen zu dürfen.

Nach einer weiteren halben Stunde verkündigte der Bärtige, dass man nun mit Sicherheit die drei ersten Teams kenne. Es überraschte niemanden, dass alle drei aus Holland stammten. Weiter sagte er noch, dass sie nicht in der Lage seien, innerhalb einer nützlichen Frist eine definitive Rangliste zu erstellen, und dass wir nun nach Hause gehen dürften. Eine definitive Rangliste werde es per E-Mail geben. Wir hatten eine Schlacht geschlagen und wussten nicht, wie sie ausgegangen war.

In dieser Ungewissheit verliessen wir das Gebäude und kamen uns gleich noch einmal "verarscht" vor. Eine halbe Stunde warteten wir vergeblich auf einen Bus, der nicht kam. So bestellten wir drei Taxis. Als sie bei der Universität vorfuhren, tauchte dann doch noch ein Mini-Bus auf, der allerdings zu klein war, um uns alle aufzunehmen.

Bei einem Kantonesen liessen wir uns kulinarisch verwöhnen. Das gute Essen und der Reiswein vermochte unsere schlechte Stimmung aufzulösen, so dass wir wieder richtig fit waren, um in einem Pub noch ein Bier zu trinken. Wir diskutierten weiter über die Idee, die europäische Ausscheidung des ACM-Programmier-Wettbewerbs nach Zürich zu bringen, die uns schon am Tag zuvor gekommen war. Nun nahm sie so richtig Gestalt an und wurde von dem einzigen Gedanken getragen, es besser als in Swansea zu machen.

Zurück im Hotel sassen die einen noch lange zusammen, während die anderen sich aufs Ohr legten, da sie am nächsten Morgen früh aufstehen wollten. Denn das eine Team hatte beschlossen, in London noch einige Stunden zu verbringen, bevor wir alle zurückflogen.

Der Tag danach

Am Montag trudelten die Langschläfer spät beim Frühstück ein. Während wir bei der Reception auf die letzten warteten, gesellte sich Ruth zu uns, und Frank und Hans besprachen mit ihr die konkreten Bedingungen für eine europäische Ausscheidung in Zürich. Der langhaarige Pseudo-Organisator lag daneben in einem Sessel und döste vor sich hin. Der Wettbewerb oder seine Organisation schien ihn erschlagen zu haben. Von einer Rangliste wusste er nichts.

Im Zug nach London wurden wieder die Monty-Python-Skripte herumgereicht. Nachdem den Rittern das "Ni!" zu langweilig wurde, begannen sie "Ekki fatang" zu sagen. Beides ist heute noch im IFW zu hören.

Im Flughafen trafen wir das Frühaufsteher-Team wieder. Sie hatten sich noch das Wachsfiguren-Kabinett und das British Museum angesehen. Zurück in Kloten wartete eine Ueberraschung auf uns: Grete war nach Kloten gefahren, um uns abzuholen und uns zu unserem Abschneiden zu gratulieren. Wie hatten wir denn abgeschnitten?

Nachgeschichte 1

Es traf wirklich noch eine Rangliste per E-Mail ein. Das Team "ETH 1" mit Patrick Aschwanden, Erich Oswald, Renato Pajarola und Manuel Bleichenbacher (das bin ich) erreichte den fünften Platz als bestes nichtholländisches Team. Wir hatten vier Aufgaben gelöst. Die besten Teams hatten fünf. Das Team "ETH 2" mit Rory Chisholm, Jörg Derungs, Daniel Friederich und Roland Ulber erreichte den sechzehnten Platz. Insgesamt haben fünfundzwanzig Teams teilgenommen (Siehe auch die Gesamtrangliste in diesem Heft).

Mit der neuen Regel hätte es gereicht für uns. Nächstes Jahr soll die Regel definitiv gelten und so sind wir alle fast sicher, dass ein Team der ETH nach Amerika gehen wird. Das sollte ein Ansporn für alle Informatik-Studenten sein, nächstes Jahr am ACM-Wettbewerb mitzumachen.

Nachgeschichte 2

Beim VIS ist man recht mail-freudig. So hat der Vorstand die Rangliste als Erfolgsmeldung aufgefasst und in der ganzen ETH herumgemailt. Ein Professor unserer Abteilung soll sich dann erkundigt haben, welche Programmiersprache die Teams der ETH verwendet haben. Meine Antwort darauf wäre gewesen: "Pascal and Modula-2 are dead." Ähnliches hört man, wenn an der ETH für Oberon geworben wird. Für den "Quick' n'Dirty"-Ansatz, der bei einem solchen Wettbewerb gefragt ist, war C wie massgeschneidert.

Die Reise nach Swansea war ein schöner Preis für den Sieg in Zürich. Wir haben viel erlebt. Noch weiss ich nicht, ob ich das nächste Jahr wieder teilnehmen oder ob ich beim Organisieren helfen soll. Dabei bin ich auf jeden Fall.

Ni!

Manuel Bleichenbacher, IIIC/7

Dank

Wir möchten uns an dieser Stelle bei folgenden Institutionen und Personen für Ihre Unterstützung bei der Durchführung der ETH-internen Ausscheidung für den ACM-Programmierwettbewerb und der Teilnahme unserer Teams an der Westeuropäischen Ausscheidung in Swansea bedanken:

Abteilung und Departement Informatik;
Prof. Zehnder,
Prof. Gutknecht;
Prof. Mössenböck,
Prof. Gonnet,
Prof. Widmayer,
Prof. Schek,
Prof. Maurer,
Prof. Nievergelt,
Prof. Mäder,
Dr. Schäuble;
Frau Hilgarth;
der Stabsstelle Software;
dem SV-Service (Polysnack),
dem Hausdienst HG
sowie unseren zahlreichen Helfern.

VIS (hd, fm)

Die Rangliste von Swansea

Auf den ersten Blick fällt die überproportional hohe Beteiligung von Teams aus den Niederlanden (12 von 25) auf. Dort ist es für die Universitäten eine Ehrensache, Mannschaften für diesen Wettbewerb zu stellen. Und das Abschneiden der jeweiligen Teams ist ein Gradmesser für die Qualität der Unis. So hat's mir jedenfalls ein holländischer Betreuer erzählt.
Angesichts dieser Tatsache dürfen unsere Resultate als sehr gut betrachtet werden. Das Team **ETH 1** mit *Patrick Aschwanden, Manuel Bleichenbacher, Erich Oswald und Renato Pajarola* belegte den ausgezeichneten fünften Platz, und das hoffnungsvolle Team **ETH 2** mit *Rory Chisholm, Jörg Derungs, Daniel Friederich und Roland Ulber* hat mit ihrem sechzehnten Rang die Mannschaften aus Deutschland, Frankreich und England auf die Plätze verwiesen. Nicht zuletzt waren sie besser als drei holländische Teams.
Qualität kommt halt immer noch vor Quantität...

(hd)

---------- Association for Computing Machinery ----------
18th International Collegiate Programming Contest
West European Region
---------- Final Standings ----------

Twenty-five teams from nine countries competed at the West European Regional Contest held at Swansea University on Sunday, 14th. November. The teams of three persons had five hours to solve a total of seven set problems. They submitted solutions written in C or Pascal.

Rank	Country	Solve	Name
1	NL	5	Rijks Universiteit Groningen
2	NL	5	Vrije Universiteit
3	NL	5	T. U. Delft
3	NL	4	University of Amsterdam
5	**CH**	**4**	**Eidgenossische Technische Hochschule Zurich**
6	NL	4	Eindhoven University of Technology
7	NL	4	Rijks Universiteit Groningen
8	S	4	University of Lulea
9	S	3	Lund University
10	IRL	3	Trinity College, Dublin
10	H	3	Technical University Budapest
12	NL	3	T. U. Delft
13	NL	2	University of Utrecht
14	SF	2	Tampere University of Technology
15	NL	2	Leiden University
16	**CH**	**2**	**Eidgenossische Technische Hochschule Zurich**
17	NL	2	Katholieke Universiteit Nijmegen
18	UK	1	University College, Swansea
19	NL	1	Eindhoven University of Technology
20	F	1	Ecole nationale superieure des mines de Paris
21	NL	1	University of Twente
22	UK	0	University of Glamorgan
22	UK	0	Staffordshire University
22	D	0	Institut fur Telematik, Karlsruhe
22	F	0	Ecole nationale superieure des mines de Paris

Der Verein der Informatikstudierenden an der ETH möchte sich bei den folgenden Firmen für die finanzielle Unterstützung

- der ETH-internen Ausscheidung für den ACM Programmierwettbewerb

und

- der Teilnahme der beiden ETH-Teams an der Westeuropäischen Ausscheidung in Swansea (UK)

bedanken:

Ascom, Bern

ABB, Baden

Digicomp, Zürich

Digitron, Aarau

Microsoft, Wallisellen

Rentenanstalt, Zürich

Chris Flu's Kochecke

Folge 12: Engadiner Kartoffelkuchen oder: Etüde in feldrosa

Nun ja, ich mag Pleoni wirklich, aber ich frage mich, ob er nicht dieses eine Mal zu weit gegangen ist. "Chris Flu", hat er gesagt, "Chris Flu, in einer Woche ist Redaktionsschluss." Naja, kann mal passieren, was geht mich das an? "No problemo", meinte er und ich dachte, nun folge das obligate Zigarettenausdrücken in meinem Bauchnabel oder eine vergleichbare filmwirksame Übersprungshandlung, aber so ist er nicht, nene, ganz im Gegenteil: das listige Auflodern seiner Augen liess mich Schlimmstes ahnen, aber erstens kommt es verheerender und zweitens als man/frau denkt. Gezielt plazierte mein Gegnüber den Bleibolzen und während er mich unvermindert freundlich anblickte, vernahm ich die unfassbaren Worte: "Hm, dann erscheinen die nächsten Visionen halt ohne Kochecke..." "Aber, morgen muss ich in den WK", blieb mir zu stammeln, dann versagte die Stimme, der Fall war klar, ich würde schreiben und wenn dies meine letzten Worte sein sollten, dem VIS will ich ewig undsoweiterundsofort.
Was nun kommt ist nicht erfunden, sondern real, live, unplugged: es ist 0347, ich sitze auf der Wache und schreibe diese Zeilen. Mehr möchte ich zu meiner momentanen Situation nicht verlauten lassen, schliesslich geben sich die Visionen in politischen Fragen relativ neutral. Punkt, fertig, no comment – aber dass der schlechte Ruf der Schweizer Armee in Bezug auf kulinarische Ereignisse nicht ganz unbegründet ist (mal abgesehen von anderen Ereignissen), muss ich im Moment am eigenen Leib erfahren. Es ist deshalb doppelt schön, mit den hier angebotenen Köstlichkeiten ein Reverseengineering zu betreiben und die erhaltenen Grundnahrungsmittel geistig zu einem olfaktorischen Gesamtkunstwerk zu verquicken. Was dabei herauskommt? Eines der schnellsten, schmackhaftesten und einfachsten Rezepte, das es gibt (drum ist es ausnahmsweise nicht von mir).

Engadiner Kartoffelkuchen

Man/frau benötigt für 1 bis 2 StudentInnen:

750 g Kartoffeln (geschält und in 1-cm^3 Würfel geschnitten)
75 g Speckwürfel
75 g Salami, in Würfel geschnitten
100 g Mehl
1/2 Kaffeelöffel Salz
1 1/2 dl Wasser

Alles vermischen, in eine ausgefettete Gratinform geben und eine Stunde bei 230 Grad im Backofen gratinieren.

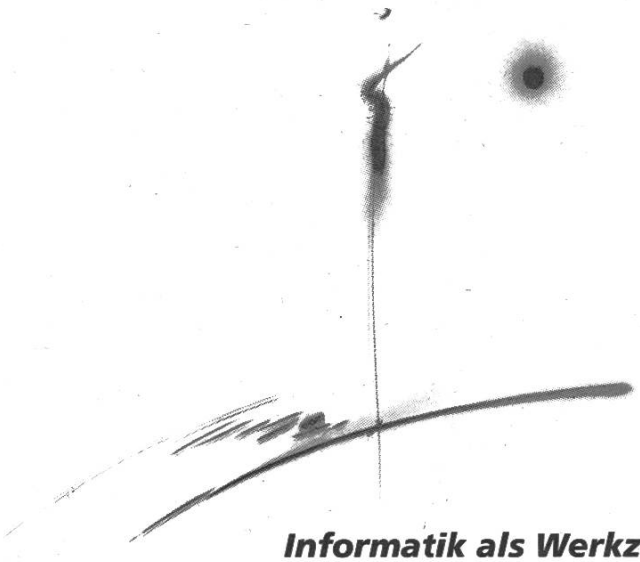
Serviert wird mit einem knackigen Salätchen. Punkto Tenu möchte ich keine Vorschriften erlassen, es empfiehlt sich jedoch, gutes Schuhwerk und zumindest einen Helm zu tragen. Es ist darauf zu achten, das Esszimmer auf -13 Grad Celsius (oder 260K) zu kühlen und vorher 2 Tage nichts zu essen, um realistische Bedingungen für die Nahrungsaufnahme zu schaffen. Zimmerordnung erstellen! Es grüsst Euch, treu dem Vaterland undsoweiterundsoetc. (siehe oben)

Chris Flu in Feldblau (sonst IIIC/8)

P.S Das Gericht ist echt lecker (kein Scherz), in dieser Kombination hab' ich's

hier natürlich nicht gekriegt, vor dem Zerlegen nannte es sich "Älplermagronen" oder so, wenn ich richtig informiert bin (ist das überhaupt möglich, informiert zu sein, da, jetzt? Keine Politik, bitte!).

Von High-Tech zu High-Sense ...



Informatik als Werkzeug. Als Instrument, das sich dem Menschen anzupassen, ihm zu dienen hat; nicht umgekehrt. Informatik zur Unterstützung der menschlichen Phantasie und zur Entfaltung schöpferischer Kräfte. Informatik für die Gestaltung eines sinnhaften Ganzen: nicht Trennung, sondern Zusammenbringen von emotioneller Wahrnehmung und technischem Verstand.

Technologische Entwicklung, Weitergeben von Know-How, betriebliche und personelle Unterstützung ... was immer die Aufgabenstellung in der Informatik ist, wir finden eine Lösung. Zusammen mit Ihnen. Denn für uns stehen die Menschen mit ihrem Denken und ihrem Fühlen im Vordergrund. Bauen Sie mit uns; bauen Sie auf uns!

SULZER INFORMATIK

SULZER INFORMATIK AG
CH-8404 WInterthur
Telefon 052 262 72 00
Telefax 052 262 72 01

1006

Accidental Empires

Unternehmen Zufall – Wie die Jungs aus Silicon Valley Millionen scheffeln, die Konkurenz schlagen und dennoch keine Frau kriegen.

Das ist der deutsche Titel eines Buches (für alle Leser, welche – unglücklicherweise – der englischen Sprache nicht mächtig sind), welches ich jedoch in der Originalfassung empfehle. Dieses Buch ist ein Bericht darüber, dass die gesamte Personal-Computer Industrie durch puren Zufall entstanden ist, und nur einen Zweck hat: Die Pubertät der involvierten Männer (Frauen kennt das Computer-Business eh nur in Nebenrollen) bis zur Senilität auszudehnen, und somit das Erwachsenwerden überflüssig machen.

Der Autor will in diesem Buch drei Dinge aufzeigen:

1. Es geschah alles mehr oder weniger durch Zufall
2. Die Leute, welche es geschehen machten, waren Amateure
3. Und sie sind es zumeist immer noch

In diesem Buch wird die Entstehungsgeschichte von Silicon Valley pointiert aufgezeigt, und dabei werden alle wichtigen und unwichtigen Personen gestreift oder getroffen.

Hier bekommt man die Antwort auf die brennenden Fragen:

Warum trägt Bill Gates Jeans und nicht Armani-Kleider?

Wer ist Mary Gates?

Was haben Saddam Hussein und Steve Jobs gemeinsam?

Wer würde ein solches Buch schreiben? Ein gefeuerter Entwickler bei Apple, Microsoft und IBM? Nein, denn diese drei sind 'mutual exclusive'.

Wer ist Robert X. Cringley? Woher weiss er, was er schreibt?

Zum ersteren: Niemand weiss, wer sich hinter diesem Pseudonym versteckt. Unter diesem Namen erscheint in jeder Ausgabe der Wochen-Computer-Zeitschrift 'Infoworld' eine Kolumne, mit dem neusten aus der Gerüchteküche, oder von NDAs[1].

Cringley gibt als Informationsquelle vor allem etwas an: Ingenieure, welche Angst haben, dass ohne Publikation ihre Erfindungen, Leistungen nie über das mittlere Management ihrer Firma hinweg zur Kenntnis genommen werden. Oder eben auch die Frustrierten, welchen genau das passiert ist.

Ein Buch für alle, die mal die menschliche Seite vom Millionen-Business im Silicon Valley kennen lernen wollen. Und ein Tröster für alle, welche (noch) kein zweiter Billy Gates sind.

Accidental Empires – How the Boys of Silicon Valley make their millions, battle foreign competition, and still can't get a date - Harper Business 1993 ISBN 0-88730-621-7 (Hardcover von Viking Books ist z.Z. Vergriffen)
Deutsch: Unternehmen Zufall - Wie die Jungs aus Silicon Valley Millionen scheffeln, die Konkurenz schlagen und dennoch keine Frau kriegen - Hardcover von Add-Wes ~48.- SFr.

[1]Non-disclosure Agreement

Lerngruppen für den Frühling!

Die nächste Prüfungssession kommt in Riesenschritten auf uns zu. Kaum ist Weihnachten vorbei, schon steht sie beinahe vor der Tür. Paaaaaaaaanik! Aber keine Angst, noch ist Zeit, diese ach so schwere Last, die wir in unseren (ja,ja auch ich gehe an die Prüfungen) Herzen tragen, einigermassen erträglich zu machen (gulb). Denn es gibt ja die

VIS-Lerngruppen

(uff)

Wenn sich genügend Leute finden, werden wir dieses Mal tatsächlich in der Lage sein, diese zu organisieren. Sie sind dazu da, den Leuten zu helfen, die gerne in Gruppen lernen. Aber auch diejenigen unter Euch, die sich schwer mit dem Lernen (oder Verstehen) tun, werden sicherlich in einer Gruppe besser auf die Prüfung vorbereitet, als wenn sie alleine lernen. Dazu kommt, dass in der Gruppe viele verschiedene Fragen auftreten: Die einen werden wohl von einzelnen Gruppenmitgliedern (Tutor?) beantwortet werden, die anderen kann man dann gezielt sammeln, um die kompetenten AssistentInnen damit zu bestürmen - Ich kann mir vorstellen, dass Alleinlernende oftmals Hemmungen haben, die AssistentInnen mit ihren Fragen zu belästigen (so à la "so wichtig wird meine Frage wohl nicht sein") - Aber gerade diese Fragen können lebensrettend sein, wenn es darum geht, ob ihr vielleicht das gesamte Kapitel nicht verstanden habt. In den Lerngruppen ist das Risiko, dass Euch so etwas passiert, sehr viel geringer! In der Gruppe übersieht man auch nicht so schnell einzelne Kapitel (was eine grosse Gefahr darstellen kann, wenn man alleine lernt – vor allem wenn die Vorlesung schlecht ist).

Ein Wort an Leute, die im Frühling nicht an die Prüfung gehen: Schön wäre, wenn gerade DU Dein Wissen in leitender Funktion einer solchen Gruppe zur Verfügung stellen würdest: Hier hast DU Gelegenheit, Deine didaktischen Fähigkeiten einmal unter Beweis zu stellen.

Meldet Euch! Es ist wirklich eine gute Sache und gar nich schwer! Ihr müsst dazu nichts anderes tun, als den Anmeldetalon (unten) auszufüllen und im VIS-Büro abzugeben.

Bis dann
lj

Anmeldung

Ja, ich bin bei einer VIS-Lerngruppe fuer die Session im Fruehling 1994 dabei

einfach als Teilnehmer ☐ als Leiter einer Gruppe ☐

Folgende Bereiche der Informatik interessieren mich:

1. VD	2. VD
☐ Informatik I+II	☐ Informatik III+IV
☐ Algebra I+II	☐ Theor. Informatik I+II
☐ Analysis I+II	☐ Wiss. Rechnen I+II
☐ Elektrotechnik I+II	☐ Elektrotechnik III+IV
☐ W'rechung & Statistik	☐ Physik I+II

Name: .....

Adresse:

.....

Telefon: .....

e-mail (waer schon toll):.....

[Red.: Äh, Leo, ich habe Dich da richtig verstanden: Die Leute sollen etwas **aus den Visionen 'rausschnippeln???]**

C-Kurs Teil 1 – Grundlagen

von Leonhard Jaschke und Patrick Leoni

Einleitung

Tja, nun ist es wohl so weit. Heute erhaltet Ihr Eure erste Lektion in "C". Was genau ist C? Und wer steht hinter C?

C wurde anfangs der 70er Jahre von Brian Kernighan und Dennis Ritchie erfunden, mit dem Ziel, das Betriebssystem UNIX in einer Hochsprache (neu-) zu schreiben – *"an unheard of step at the time"* [2] – vorher gab es für Betriebssystemprogrammierung nur Assembler. Die beiden veröffentlichten ihr Werk "The C Programming Language" (zu deutsch "Programmieren in C", [1]) im Jahr 1978, und die Sprache machte Furore. Sie wurde ziemlich beliebt (schwierig nachzuvollziehen :-)), und verschiedene Compilerbauingenieure änderten ziemlich viel daran herum (das ist leichter verständlich...). Im Jahr 1988 kam dann das ANSI und stellte einen Standard vor, das fortan ANSI C genannte genormte C. Die beiden Erfinder schrieben eine neue Version ihres sehr erfolgreichen Buches, und dieses heisst im Untertitel "Second Edition, ANSI C" bzw. "Zweite Ausgabe, ANSI C". In den fünf Jahren seit der Vorstellung der neuen Ausgabe von C hat sich einiges insoweit geändert, als die meisten erhältlichen C-Compiler ANSI C verstehen – aber den alten Code nach wie vor durchgehen lassen. Niemand kann sich heute auf das Verständnis von ANSI C beschränken, zu viel Code ist in Standard C ("K&R-C") geschrieben – und auch heute noch wird viel in diesem alten, definitionsgemäss eher "schlampigen" Stil codiert. Dennoch: Wir sollten uns befleissigen, nur noch die neue Art zu programmieren zu gebrauchen. Auch in diesem Kurs werden wir versuchen, alle *non-ANSIsms* zu vermeiden.

Jetzt geht's los. Was haben wir mit Euch vor? – Dieser Teil wird Euch zunächst mit den grundlegenden Kontrollstrukturen und Statements von C vertraut machen, wir werden Euch zeigen, wie ein C-Programm aussieht, wie man Variablen deklariert, was für grundlegende Typen C kennt ... – All das soll uns helfen, noch heute ein kleines Programm zu erarbeiten und es zu compilieren. Dieses Programm, ein kleines Sortierprogramm für Zahlen, wird am Ende dieses ersten Teils dastehen. Wie man das Programm dann professionell debuggt, werden wir erst nächstes Mal beschreiben (Ceres-Hacker wissen ohnehin, wie man ohne Debugger zurechtkommt). Also gehen wir es an:

Prinzip

C ist eine imperative Programmiersprache. Das heisst: In C teilen wir dem Computer mit, was er wann machen soll, damit wir das bekommen, was wir von ihm wollen – statt dass wir das beschreiben, was wir wollen und es dann dem Computer überlassen herauszufinden, wie er das vollbringen soll (wie in anderen Sprachen). So wie in Oberon wird in C zwischen Gross- und Kleinschreibung unterschieden. Entgegen den Brauch in Oberon werden allerdings hier die Schlüsselwörter alle klein geschrieben.

Der Aufbau eines typischen C-Programmes

Grob gesagt kann man in einem C-Programm zwei Teile unterscheiden: Den Deklarationsteil und den Funktionsteil. Diese beiden Teile sind nicht notwendigerweise disjunkt (d.h. man kann im Funktionsteil deklarieren, man kann auch im Deklarationsteil Funktionen schreiben), aber es hat sich als günstig herausgestellt, diese beiden Teile zu trennen.

Im Deklarationsteil werden folgende Arbeiten erledigt: Importieren von externen Objekten (Funktionen, Typen, Variablen), Deklaration von Konstanten, Typdefinitionen, globale Variablendefinition.

Im Funktionsteil definiert man dann die Funktionen und Prozeduren.

Kommen wir zuerst auf den Deklarationsteil zu sprechen.

Elementare Datentypen

Bevor wir die Variablendeklaration genauer anschauen können, müssen wir zuerst noch auf die Grundtypen dieser Sprache zu sprechen kommen. Ich möchte hier ein Liste der Typen geben, die Ihr von Pascal ff. schon kennt. In dieser Liste seht ihr links für jeden Typ ein Euch bekanntes Schlüsselwort, rechts steht die *ungefähr* entsprechende C-Version des Typs:

```
SHORTINT    short
INTEGER     int
LONGINT     long

CHAR        char

REAL        float
LONGREAL    double
```

Das sind also die Grundtypen in C, die mit den Pascaltypen eine Entsprechung haben. Aus den Ganzzahlen lassen sich durch Voranstellen des Schlüsselworts *unsigned* natürliche Zahlen machen (dann entspricht eine der Typ *unsigned int* dem in Modula existierenden Typ *CARDINAL*).

Wie Ihr vielleicht bemerkt habt, haben wir dem Datentyp *BOOLEAN* keine Entsprechung in C gegeben. Er ist tatsächlich kein Grunddatentyp und wird durch eine Partition der Integer-Zahlen "simuliert": In C hat *FALSE* den Wert 0 und *TRUE* einen beliebigen anderen Wert. Wenn Ihr mit boolschen Variablen arbeiten möchtet (aber wirklich nur dann! siehe unten), solltet Ihr Euch am Anfang Eures Programmes einen Datentyp *BOOL* und die beiden Konstanten *FALSE* mit Wert 0 und *TRUE* mit Wert 1 definieren. Dies funktionert mit #define (siehe unten). Man kann damit gut rechnen, und man sieht, was man meint.

Variablendefinition

Bevor wir uns den ersten zusammengesetzten Datentyp (den Array) anschauen, möchte wir Euch noch die Variablendefinition ans Herz legen. Eine Variablendefinition hat folgenden Aufbau:

Typenname Variablenname;

Anstelle des Variablennamens können auch mehrere durch Komma getrennte Namen stehen. Typische Beispiele für eine Variablendefinition wären also:

```
int i,j;
char ch;
long x;
```

Es ist möglich, eine Variable gleich bei der Definition zu initialisieren. Das geht so:

```
int pos= 69;
```

Wichtig ist: Es existiert kein Schlüsselwort für die Variablendeklaration, einzig und allein der Typenname entscheidet, dass es sich hier um eine Variablendeklaration handelt (wie Ihr später sehen werdet, fangen allerdings auch Funktionsdeklarationen mit einem Typennamen an).

Arrays

Jetzt, wo wir wissen, wie eine Variable deklariert wird, können wir das ganze so erweitern, dass wir dem Compiler mitteilen, dass die deklarierte Variable eigentlich ein Vektor mit mehreren Komponenten ist. Dies geschieht so:

Typenname Variablenname[Anzahl Komponenten];

Im Grunde ist diese Deklaration (wenn auch anders geschrieben) mit der von Oberon vergleichbar. Wie in Oberon werden auch in C die Arrays von 0 an indiziert. Das letzte Element hat daher auch hier den Index (Anzahl Komponenten)-1. Ein einzelnes Element wird angesprochen, indem man in eckigen Klammern den Index des Elementes hinter den Variablennamen schreibt.

Beispiel:

```
int Vektor[5];
```

Mit Vektor[3] bekommen wir das vierte Element dieses Vektors. Vektor[5] ist nicht zulässig, da der Vektor nur die Elemente mit Index 0,1,2,3 und 4 besitzt. Vektor[5] referenziert also ein Element, das nicht existiert. So etwas kann zu unvorhergesehenen Abstürzen führen. Vorsicht, die wenigsten C-Compiler fangen einen solchen Fehler ab.

Ein mehrdimensionaler Vektor (also ein *Array*) wird durch mehrere Klammerungen definiert. Die Zeile

```
int Matrix[3][4];
```

definiert uns zum Beispiel eine 3x4-Matrix. Man kann mit *Matrix[2]* die zweite Zeile der Matrix ansprechen. *Matrix[2]* ist ein Vektor mit 4 Elementen. Leider kann man die Spalten nicht so einfach referenzieren.

Die Vorbelegung eines Arrays geht auch, und zwar wie folgt:

```
float Noten[]=
{4.47, 4.69, 4.66, 4.63}
```

Die nötige Anzahl floats findet der Compiler selbst heraus. Es geht auch, zwischen [] eine Zahl hinzuschreiben, dann wird so viel Speicher alloziert wie angegeben. Bei mehrdimensionalen Arrays läuft das so:

```
int TicTacToe[3][3]=
{{0,1,1},{-1,1,0},{-1,-1,0}}
```

oder, etwas unübersichtlicher:

```
int TicTacToe[3][3]=
{0,1,1,-1,1,0,-1,-1,0}.
```

Konstanten

Es macht sich immer gut, in seinen Programmen keine *Magischen Zahlen* stehen zu haben, also etwa eine 13.25 irgendwo mitten in einem Ausdruck mit Variablen. Was diese Konstante heisst, ist im Mittel 2.34 Tage nach der Programmerstellung

vergessen. Ausserdem wird sie meist an anderen Stellen im Programm auch noch gebraucht, und so liegt es nahe, hier eine regelrechte Konstante zu definieren. Wenn diese auch noch einen guten Namen bekommt (also etwa "NotensummeVD93" oder so), dann sieht ein Programm schon viel professioneller aus. Dies geht in Pascal mittels CONST, in C heisst das:

```
#define NotensummeVD93 13.25
```

Jedes Auftreten von "NotensummeVD93" im Programmtext wird vor dem eigentlichen Compilerdurchlauf dann automatisch durch die Zahl ersetzt.

Statements

Kommen wir zu den elementaren Anweisungen, die in jeder imperativen Sprache vorhanden sein müssen:

Die Zuweisung

Wir sind der Zuweisung eben begegnet, bei der Initialisierung. Genauer nimmt sie in C folgende Erscheinungsform an:

Variable= Ausdruck;

Zwei Dinge sind hier zu beachten (weil sie uns nicht so vertraut sind): Erstens: Die Zuweisung sieht dem Vergleichsoperator **verdammt ähnlich** und wird auch relativ oft mit ihm verwechselt, hat aber eine andere Semantik (dummerweise kann das der Compiler oft gar nicht erkennen). Zweitens: **Jede** Anweisung wird mit einem Strichpunkt abgeschlossen (auch die, die am Ende eines Blocks (s.u.) steht).

Der oben erwähnte Ausdruck kann natürlich auch aus einer einzigen Konstante bestehen. Die meisten konstanten Ausdrücke sind der Oberon-Notation gleich. Character-Konstanten werden allerdings als solche mit Apostrophen gekennzeichnet:

```
char ch;
ch= 'a';
```

Float- oder double-Konstanten müssen nach der alten Kernighan & Ritchie- Konvention ein Komma enthalten:

```
float f;
f= 1.0;
```

Nach der neuen ANSI-Konvention kann man das Komma schreiben, muss aber nicht.

Ich möchte darauf hinweisen, dass auch Folgendes in C möglich ist:

```
a=(b=(c=2)*5)*4;
```

Diese Zeile besteht eigentlich aus drei Anweisungen:

```
c=2; b=c*5; a=b*4;
```

Ich rate jedem davon ab, mit diesen Mehrfachzuweisungen, wie man sie euphemistisch nennt, zu operieren (auch wenn es in C ein Sport zu sein scheint, Programme in dieser Weise abzukürzen). Sie fördern die Verständlichkeit der Programme in keiner Weise. Eventuell werdet Ihr allerdings einmal in die Lage kommen, C-Programme von anderen Leuten zu lesen (die diesen schlechten Stil auch noch toll finden). Dann sollt Ihr nicht unbedingt vor den Kopf gestossen sein und nicht wissen, was das bedeutet. Aber begeht um himmels Willen nicht selbst solche Verbrechen!

An dieser Stelle stellt sich nun wohl die

Frage: Was darf ich welchem Variablentyp zuweisen? Diese Frage ist heikel... Beginnen wir damit, wie es sein sollte: Jeder Variable kann man natürlich Werte ihres eigenen Typs zuweisen. Ausserdem gibt es – wie in Oberon – Typhierarchien:

double>float>long>int>short (=char)

Also: Einer long-Variable darf man einen int-Wert zuweisen, einer double-Variable einen float-Wert.

In der anderen Richtung geht es etwas schwieriger: Da muss man dem Compiler mitteilen, in welchen Typ man konvertieren will. Das macht man, indem man die Typenbezeichnung in Klammern vor den zu konvertierenden Ausdruck schreibt:

```
float a;
int c;

a= 1.0;

c= (int)a;
c= (int)1.0;
c= (int)(a+1.0);
```

Diese Art von Typkonvertierung nennt man *explizites casting*, die oben beschriebene automatische Typenkonvertierung bezeichnet man als *automatisches casting*, und die Typenbezeichnungen in Klammern nennt man die *Cast-Operatoren*. Das casting in C entspricht in seiner Gefährlichkeit (und Mächtigkeit) ungefähr der VAL-Funktion in Oberon. Mit einem kleinen Unterschied: Die VAL-Funktion in Oberon habt Ihr vielleicht noch gar nicht registriert, in C allerdings werdet Ihr noch oft casten.

Ausdrücke

Nur beiläufig wollen wir die Ausdrücke erwähnen. Diese sind nämlich, wie gesagt, sehr ähnlich wie in Pascal ff. Die grundlegenden Operatoren sind sogar ganz gleich:

+ plus, – minus, * mal und / geteilt durch.

% entspricht MOD und kann nicht auf float-Werte angewandt werden.

Das / entspricht bei zwei int-Operanden einem Operon-DIV, bei mindestens einem (Stichwort *automatisches casting*) float einer Gleitkomma-Division.

In C gilt, fast überflüssig zu erwähnen, die alte *Punkt-vor-Strich*-Regel.

Eine C-Spezialität sind die Inkrement- und Dekrement-Operatoren. Diese werden ziemlich häufig angewendet und funktionieren wie folgt: Die Statements

```
c++ sowie ++c
```

erhöhen beide die Variable c um eins. Der Unterschied ist, dass mit

```
int c=0
```

a nach der Zuweisung (!)

```
a= c++
```

eine 0, aber nach

```
a= ++c
```

eine 1 erhält. Die Stellung des Operators entscheidet also darüber, ob an a das Ergebnis der Berechnung vor oder nach der Auswertung des Operators ++ zugewie-

sen werden soll. (Den Unterschied nennt man *Post-* bzw. *Prefix-Notation*). Gleiches gilt für – –.

Abkürzungen für Arithmetische Ausdrücke

Die Zuweisung

```
i= i + 2;
```

kann mittels

```
i+= 2
```

kürzer geschrieben werden.

Dies funktioniert auch für die anderen arithmetischen Operatoren, also für

```
+, -, *, / und %
```

sowie für ein paar Bit-Manipulationen (die werden später erklärt)

Kontrollstrukturen

Die anderen elementaren Konstrukte kennt ihr von Oberon her. Ich möchte hier eine Tabelle angeben, die Euch hilft, diese Konstrukte nach C zu übertragen:

```
IF Bedingung THEN
  Statementsequence
  Statementsequence

FOR var:= from TO to DO
  Statementsequence
END

WHILE Bedingung DO
  Statementsequence
END

REPEAT
   Statementsequence
UNTIL Bedingung

CASE Variable OF
 val1: Statementsequence
|val2: Statementsequence
.
.
ELSE
Statementsequence
END
```

```
if ( Bedingung )
  { Statementsequence; }
else
  { Statementsequence; }

for (var=from; var<=to; var++)
 { Statementsequence; }


while (Bedingung)
 { Statementsequence; }


do
 { Statementsequence; }
while (!Bedingung)

switch(Variable)
 {
   case val1: Statementsequence;
              break;
   case val2: Statementsequence;
              break;
    .
    .
   default:   Statementsequence
 }
```

Zunächst zum switch: Es ist unerlässich, dass jede Alternative mit einem break-Statement abgeschlossen wird. Sonst werden einfach die Alternativen durchlaufen. C.
Dann ein Wort zum for: diese Kontrollstruktur ist ein Multikulti, sie kann mehr als zum Beispiel in Oberon(-2...) die FOR-Anweisung. In Initialisierung im ersten und in Iteration im dritten Teil des Schleifenkopfs können mehrere Ausdrücke, durch Kommas getrennt, angegeben werden. Damit kann man manchmal den ganzen Schleifenrumpf in den dritten Teil des Schleifenkopfs verpacken, und der Rumpf bleibt leer. Besonders stilvol ist das allerdings nicht. Aber über Geschmack, nicht wahr,...
Achtung: Die Bedingung, also der zweite Teil des Kopfs ist als *solange-als-gilt-Ausdruck* zu verstehen, nicht als *bis-dass-gilt*.

Wie sehen nun die Bedingungen in C aus? Wie oben schon angedeutet, werden boolsche Ausdrücke in C mit Ganzzahlen simuliert. Dennoch gibt es eine Reihe von boolschen Operatoren, in C allerdings "Vergleichs- und logische Verknüpfungsoperatoren" genannt:

==	Gleichheitsoperator
!=	Ungleichheitsoperator
<	Kleiner-Operator
>	Grösser-Operator
<=	Kleiner-Gleich-Operator
>=	Grösser-Gleich-Operator
&&	Logisches UND
||	Logisches ODER
!	Logische Negation

Also: Eine Bedingung, die in Oberon so aussieht

```
(A="A") & (B#"B") & ~(C<=13)
OR (D>4)
```

nimmt in C folgende Erscheinungsform an:

```
((A=='A') && (B!='B') &&
!(C<=13)) || (D>4)
```

Auch in C bindet ein && stärker als ein ||, aber damit der Compiler keine Warnungen absondert, sollte man seine Präferenzen mittels Klammern akzentuieren. Die Fehleranfälligkeit wird so verkleinert.
Ja, so schwer ist das nicht, wenn man einmal die Notationen gelernt hat.

Noch etwas: Eigentlich sind Bedingungen ganz gewöhnliche Ausdrücke. Dies folgt schon daraus, dass ein Wahrheitswert ja nicht irgendein BOOLEAN ist sondern durch ints simuliert wird.

Illustrieren will ich das mit dem Problem des Rosenschenkens: Einem Mann Rosen zu schenken ist nicht so gebräuchlich. Dies aber bei einer Frau zu tun, schon eher. Wenn ich also die Funktion strcmp() (String compare) verwenden will [Exkurs: strcmp(string1, string2) ist eine Funktion, die 0 zurückgibt, falls die beiden Strings *gleich* sind, und sonst irgendeine Zahl != 0], so *darf ich nicht* sagen:

```
if ((strcmp(Person,"Mann") =
TRUE)) SchenkeRosen;
```

Denn TRUE ist ja eine Konstante (=1, s.o.), und kann also den vielfältigen Rückgabewerten von strcmp() nicht gerecht werden.
Wenn ich (nur) einer *Frau* Rosen schenken will, ist folgendes richtig (und gebräuchlich):

```
if (strcmp(Person,"Mann"))
SchenkeRosen
```

Wenn der Vergleich eine Zahl != 0 liefert, die Person also *kein* "Mann" (also, sa-

gen wir einmal, eine "Frau") ist, so wird dieses Resultat gleich als Wahrheitswert gebraucht. Und da jede Zahl != 0 *wahr* bedeutet, gibt's Rosen.
In der ersten Fassung wäre bei einer "Frau" zwar auch strcmp() != 0 herausgekommen. Es wäre aber Zufall gewesen, wenn dieses Resultat gleich 1 gewesen wäre, der Vergleich mit TRUE hätte also höchstwahrscheinlich fehlgeschlagen.

Kommentare

Was in Oberon die (* *) sind, sind in C die /* */. Dazwischen stehen Kommentare, die der Compiler gar nicht erst zu interpretieren versucht. Kommentare dürfen in C nicht verschachtelt werden.

Blöcke

Anweisungen, also etwa Zuweisungen und Funktionsaufrufe (kommt noch) können gruppiert werden. Mit {} lassen sich Statements zusammenfassen, genauso wie das mit BEGIN-END in Modula, oder mit der Statementsequence in Oberon möglich ist – dort braucht es ja nicht so exzessive BEGIN-Orgien. Mit solchen geschweiften Klammern gruppierte Statements lassen sich syntaktisch wie ein einzelnes Statement verwenden.

Konsolenausgabe

Wer etwas berechnet hat, will es dann auch ausgeben. Eine Ausgabeprozedur wie WriteLn ist hierzu ganz nützlich – in C heisst das printf(). printf() gibt einen String auf die Standardausgabe aus. Diese ist im Normalfall einfach die aktuelle Konsole. Es braucht anders als in Modula-2 oder Oberon nicht viele verschiedene Versionen dieses Befehls – in Oberon muss zur Ausgabe der Inhalte von zum Beispiel 5 Variablen gleichen oder verschiedenen Typs fünfmal eine Ausgabeprozedur wie Out.Int oder Out.Real aufgerufen werden. In C kann alles mit einem einzigen Aufruf der Funktion printf() geschehen – das heisst so etwas wie print formatted. Da in C Argumentlisten anders als in den Pascal-ähnlichen Sprachen nicht unbedingt feste Länge haben müssen, ist printf() genauso zur Ausgabe eines kurzen Meldestrings wie fürs Drucken einer längeren Liste von Daten aufs Mal geeignet.

Wie funktioniert nun dieses printf()? Ziemlich einfach. Es erwartet einfach als Mindesteingabe einen Formatstring, der dann das Format der ganzen Ausgabezeile festlegt. Das sieht dann etwa aus wie folgt:

```
printf("Dies ist ein Formatstring.\n");
```

Diese Zeile gibt, wie nicht anders zu erwarten, die Meldung aus:

```
Dies ist ein Formatstring.
```

Und was ist mit dem \n? n wie New-Line? Genau. So ein \n fügt an der entsprechenden Stelle in der Ausgabezeile einfach einen Zeilenumbruch ein. Da dieser je nach Betriebssystem anders definiert ist ("War es nun 10 oder 13 oder nicht doch 10 gefolgt von 13...?"), ist dieses \n hübsch portabel.
Genauso wie \n existieren noch diverse andere Steuerzeichen.
Wichtig sind darunter nur die paar folgenden:

`\a`	Klingelzeichen
`\b`	Backspace
`\f`	Form Feed
`\r`	Carriage Return (Wagenrücklauf, ohne Zeilenvorschub)
`\t`	Tabulator
`\xhh`	hexadezimale Zahl, Ausgabe als ASCII-Zeichen

Ein paar Zeichen können im Formatstring nicht so ohne weiteres aufgeführt werden, denn sie sind für irgendwelche Spezialzwecke vorgesehen. Dies ist das Fragezeichen (?), der Apostroph (´) das Anführungszeichen (") sowie, natürlich, der Backslash (\) selbst. Diesen Zeichen muss ein \ vorangehen (z.B. \\).
Der Formatstring enthält zusätzlich vor allem aber sogenannte Umwandlungszeichen. So kann man angeben, wo im Formatstring ein weiteres Argument ausgegeben werden soll:

Mit

```
int i = 42;
```

ist die Ausgabe von i mittels

```
printf("i = %d, man glaubt es
kaum...", i);
```

möglich. Das Argument i wird nun in einen String umgewandelt und ausgegeben. Das Resultat ist eine Ausgabe à la

```
i= 42, man glaubt es kaum...
```

So wird also ein Integer ausgegeben. Genau gleich ist es möglich bei Zeichen (char), reellen Zahlen (float), Zeichenketten etc. Die entsprechenden Umwandlungszeichen lauten wie folgt:

`%d` oder `%i`	int
`%x` oder `%X`	Ausgabe eines int als hex bzw HEX-Zahl.
`%c`	Zeichen, ein Character
`%%`	Ein Prozentzeichen wird ohne Umwandlung ausgegeben.
`%s`	Eine Zeichenkette, also ein Vektor mit Characters (kommt noch)
`%f`	float

Soviel zur Ausgabe. Die Ausgabe in ein File funktioniert ziemlich gleich, aber da Files im Gegensatz zum Bildschirm vor Gebrauch geöffnet werden müssen, gestaltet sich dort die Sache ein wenig langwieriger. Wir besprechen Files dann in einer der nächsten Folgen.

Hauptprogramm

Wie in Pascal und Konsorten gibt es auch in C Unterprogramme, sie heissen "Funktionen". Dieser Name trifft zumindest in den Augen des Pascal-Programmierers durchaus zu, denn C-Funktionen geben bei ihrer Beendigung immer einen Wert zurück. Dieser kann allerdings ignoriert werden, und eine Funktion kann auch einen ungültigen (void) Wert zurückliefern der dann überhaupt nicht verwendet werden kann. Wichtig ist hier allerdings noch nicht das Konzept mit den Funktionen, sondern dass es ebenso wie in Pascal ff. ein Hauptprogramm gibt, bei dem ein Programm startet. Dieses sieht aus wie eine Funktion:

```
main().
```

In Wahrheit *ist* main() eine Funktion. Es erhält nämlich als Parameter vom Betriebssystem das übergeben, was auf der Programmzeile beim Aufruf des Programms folgt also die Argumente des Programms. Einer Weiterverarbeitung ist so keine Schranken gesetzt. Die volle Schreibweiseder obigen Zeile lautet also

```
void main(int argc, char
*argv[]);
```

Das "void" gehört in ANSI dazu, sonst ernten wir eine Warnung vom Compiler von wegen ANSI-Verletzung.
argc enthält die Anzahl der auf der Kommandozeile übergebenen Worte, der Zei-

chenkettenvektor die Argumente selber. Da Zeichenketten später besprochen werden, sollten diese Angaben reichen.
Zurück zu main(). Wenn in Oberon das kürzeste Programm

```
MODULE Kurz; END Kurz.
```

sowie in Pascal

```
PROGRAM Kurz; BEGIN END.
```

lautet, heisst es in C einfach

```
main() {}.
```

Getrennte Übersetzung

Was soll ich schon von *getrennter* Compilation reden, wo ich doch nicht einmal die Compilation *an sich* erwähnt habe...? Ganz einfach: Die erwähnten Funktionen printf() und strcmp() gibt es natürlich bereits in vorcompilierter Version. Wir müssen nicht jedesmal, wenn wir etwas ausgeben wollen, zuerst das printf() programmieren oder zumindest abtippen. Klar. Dafür gibt es die getrennte Übersetzung. Wir können also wie in Oberon gewisse Funktionen importieren (dort heisst das IMPORT). Wir bewerkstelligen das in C mittels:

```
#include
```

Und wenn wir printf() benutzen wollen, ist ein

```
#include "stdio.h"
```

nötig. Dann wird die sogenannte Headerdatei von stdio eingebunden (Genaueres ein anderes Mal), und im "Modul" stdio sind viele nützliche I/O-Funktionen versammelt. Die Funktion strcmp() ist zum Beispiel mittels

```
#include "strings.h"
```

zugänglich. Dass printf() auch ohne #include "stdio.h" funktioniert, wollen wir hier grosszügig übersehen. Das ist allerdings nicht ein *Vorzug* von C, sondern ein Mangel (zumindest in unseren Augen). Der Linker findet halt auch ohne #include jene Funktionen, die er dazulinken muss.
C, you know.

Der Höhepunkt

Wir sind (endlich!!!) am Ende der heutigen Folge. Es war ein ziemliches C-Bombardement, ich gebe es zu, aber die Rosine kommt noch: Ein Programm, in C, das etwas... *tut*!

Was Ihr zu tun habt ist nur das folgende:

Eintippen, compilieren und dann... watch it fly!
Was es tut und wie es das tut? Na, selbst herausfinden...!

Also – Voilà:

```
/******* Visionaerer C-Kurs *******/
/* Programm 1:                     */
/* Dieses Programm sortiert den    */
/* Integer-Array a aufsteigend.    */
/* Autor: Leonhard Jaschke, unter */
/* Einfluss von Patrick Leoni      */
/* Datum: 28.11.1993,              */
/* 16:15 - 16:25 Uhr               */
/*********************************/

#include "stdio.h"

#define N (10)
int a[]= { 10,7,2,8,1,5,6,9,4,3 };

void main()
{
  int i,j;
  int x;

  for(i= 1; i<N; i++) {
    j= i;
    x= a[j];
    while((j!=0) && (x<a[j-1])) {
      j--;
      a[j+1]= a[j];
    }
  a[j]= x;
  }
  for(i= 0; i<N; i++)
    printf("%d\n",a[i]);
}
```

Noch was: Eintippen könnt Ihr die Sache mit vi, emacs, textedit oder einem anderen auf der Sun verfügbaren Editor. Speichert das Programm unter

```
sort.c
```

Compilieren könnt ihr auf der Sun mittels

```
gcc sort.c
```

und das erzeugte Programm hört auf den schönen Namen

```
a.out
```

Also: Mit a.out könnt ihr das Programm starten.
gcc ist der GNU-C-Compiler. Der standardmässig mit UNIX mitgelieferte cc wird bei Solaris, ebenso standardmässig, nicht mehr mitgeliefert. Und wohl weil die ETH sparen muss, hat man sich stattdessen den Public-Domain-gcc besorgt. Der funktioniert auch, und darum wird im Moment damit alles compiliert.
Wenn ihr ganz sicher gehen wollt, dass der Compiler alle *ANSI-C-Don'ts* reklamiert, so compiliert einfach mit

```
gcc -Wall -pedantic sort.c
```

So! Viel Spass, schöne Weihnachten, bis zum nächsten Mal

Leo und Patrick

Literatur:

[1] Kernighan/Ritchie: Programmieren in C. Hanser und Prentice Hall 1990

[2] M. Bach: The Design of the UNIX Operating System. Prentice Hall 1986

Neues aus der Abteilung für Informatik

Urlaub zwischen Grundstudium und Fachstudium
Auf vielseitigen Wunsch der Studierenden hat das Rektorat im Sinne einer Gleichberechtigung eine neue Regelung beschlossen. Wer direkt nach dem 4. Semester Urlaub (1 Semester oder gar ein Jahr) macht, der bleibt für den ganzen Urlaub im 4. Semester (parkiert), bis er beschliesst das Fachstudium anzutreten. Wer hingegen zuerst das 5. Semester belegt, und erst im folgenden Semester Urlaub macht, bei dem zählen die Semester weiter, d.h. er kommt ins 6. Semester.

Teilweise schon beim alten, und vollständig beim neuen Studienplan ist die Semesterzahl im Fachstudium absolut ohne Bedeutung für den Studieninhalt. Der einzige Unterschied zur früher festgelegten Regelung, die ich bisher verbreitet habe: Ein Urlaubsemester direkt nach dem Grundstudium wird nicht mehr als Studiensemester betrachtet und hat keinen Einfluss auf die 8 Semester Maximalstudiendauer im Fachstudium.

Achtung: Wer ein Urlaubssemester zwischen Grundstudium und Fachstudium einschiebt, kann keine Vorlesungen aus dem Fachstudium besuchen, auch nicht Ergänzungen und Anwendungen. Alle gegenteiligen Auskünfte meinerseits beruhten auf der alten Regelung.

Robotik-Vorlesungen
Bisher (im Semesterprogramm bis und mit WS93/94!) waren die Robotik-Vorlesungen mangels einer besseren Möglichkeit jeweils unter "Informatik und Anwendung" aufgeführt und somit für Informatik-Studierende allenfalls als Nebenfachvorlesung mit Bewilligung wählbar. Als Vertiefungsvorlesungen waren sie nicht wählbar, und ein Bereich "Informatik und Anwendung" war im alten Studienplan 89 nicht vorgesehen. Wer nach altem Studienplan die Robotik-Vorlesungen unter "Informatik und Anwendung" besuchte, bekam bestenfalls ein Testat dafür. Angerechnet wurde nichts.
Da die Robotik nicht den Anforderungen des neuen Studienplanes für die Kategorie "Anwendung" entspricht, diese Vorlesungen jedoch als Nebenfach sehr beliebt sind, wurde das Standard-Nebenfach "Robotik" geschaffen. Die Robotik-Vorlesungen sind somit auch nach neuem Studienplan nur in der Kategorie Nebenfach zugelassen.

An die Studierenden nach altem Studienplan 1989:
Professor Mössenböck verlässt die ETH Ende Wintersemester 1993/94 und nimmt in diesem Frühjahr letztmals Prüfungen ab. Im Einverständnis mit dem Abteilungsvorsteher und dem Prorektorat für Diplomstudien dürfen also Studierende die Fächer

Objektorientierte Programmierung
Softwaretechnik
Compilerbau II

vorgezogen in diesem Frühjahr prüfen lassen, auch wenn sie die Prüfungsstufe im Verbund erst im Herbst 1994 oder längstens im Herbst 1995 (Auslauf des alten Studienplanes) ablegen möchten. Die Noten werden dann beim Abteilungssekretariat gespeichert und zum gegebenen Zeitpunkt in das Prüfungspaket des 2. Teils des Schlussdiploms eingebaut.
Beim Vorziehen der genannten Prüfungen im Frühjahr 1994 ist eine Anmeldung beim Rektorat **nicht** nötig, wohl aber eine Meldung an das Abteilungssekretariat unerlässlich.

An die zum neuen Studienplan Uebertretenden aus höheren Semestern:
Zum neuen Studienplan Uebertretende, die die Vorlesungen vor dem Inkrafttreten des Studienplanes 1993 besucht haben und diese nun nach der Uebergangsregelung noch im Frühjahr 1994 bei Professor Mössenböck prüfen lassen möchten, müssen sich natürlich regulär beim Rektorat anmelden.

Bitte nicht vergessen:

Letzter Anmeldetermin für die Prüfungen Frühjahr 1994:

4. Januar 1994

Anmeldekarten können ausser der dafür festgelegten Zeit 1.-10. Dezember, längstens bis und mit 23. Dezember 1993 beim Abteilungssekretariat abgeholt werden. Ab 24. Dezember bis und mit 3. Januar ist das Sekretariat geschlossen.

Mit den besten Festtagswünschen

Abteilungssekretariat

H. Hilgarth und L. Perrochon

YOUR FUTURE IS OUR FUTURE

Logica is a leading international computer systems integration, software and consultancy company. We have been involved in some of the key developments in our field - developments that have a real impact on the advancement and application of technology.

Our work revolves around the skills of our carefully selected and trained, high-calibre staff in harnessing the power of the computer to meet clients' needs, to enable systems to operate more effectively and efficiently.

Logica can offer you opportunities in the field of information technology. In Switzerland our emphasis is on the financial sector. Worldwide, Logica is active in all areas of information technology from space, broadcasting, transport and energy to defence, manufacturing and telecommunications.

At Logica we have a genuine commitment to career development and place great emphasis on individual initiative. We want you to help us shape the future of IT whilst gaining a sound start in a stimulating career.

To find out more contact:

Fr. Lyn Dunk
Logica Informatik AG
Schaffhauserstrasse 358
8050 Zürich

or see us at the Kontakt Party on January 24

logica

Software Engineering bei der
SWISSAIR
Ein Praktikumsbericht

Ausgangslage
Die Abteilung *Software Engineering und Datamanagement* (Team von ca. 7 Leuten, alle mit Hochschulabschluss) der Firma Swissair ist für das "Swissair Informatik Projektvorgehen" (SIPV) verantwortlich. Das SIPV ist eine auf die Swissair zugeschnittene Variante des britischen Vorgehens SSADM (Structured System Analysis and Design Method). Obwohl das SIPV fertig definiert und beschrieben ist, und die Ausbildung der Mitarbeiter für dieses Projektvorgehen läuft, hat die Swissair (noch) kein CASE-Tool, welches das SIPV unterstützt und die Entwicklungsdaten verwaltet, welche im Laufe der Projekte entstehen.
Zur Zeit meines Praktikumantritts war eine Vorstudie zum Thema Dokumentations-Umgebung für das SIPV im Gange.

Meine Tätigkeit in Kürze
Mein Praktikum bei der Swissair dauerte 14 Wochen. In der ersten Woche konnte ich als Einstieg an einem viertägigen SIPV-Kurs teilnehmen. Anschliessend erhielt ich die Aufgabe, das SIPV auf einem CASE-Tool abzubilden und anhand der Kursfallstudie die Tauglichkeit dieser Software ein erstes Mal grob abzuklären. Parallel dazu arbeitete ich an der obenerwähnten Vorstudie mit und liess meine Erfahrungen mit dem CASE-Tool einfliessen.
Aufgrund der in der Vorstudie präsentierten Fakten und Schlüsse wurden zwei Pilotprojekte bewilligt und das CASE-Tool (Software-Lizenzen und Hardware) für ein halbes Jahr gemietet. Im Rahmen dieser Pilotprojekte trug ich einen wesentlichen Anteil zur Bereitstellung des CASE-Tools bei und übernahm die erste Mitarbeiter-Schulung.

Etwas detaillierter...
Im firmeninternen Kurs, den ich am Anfang des Praktikums besucht habe, wurde den Mitarbeitern das SIPV mit seinen Techniken und Abläufen anhand einer Fallstudie vermittelt. Der Kurs war sehr gut vorbereitet und didaktisch recht hochstehend. Ich konnte in diesen vier Tagen viel profitieren. Vor allem wurde mir zum ersten mal richtig bewusst (obwohl ich schon vorher ein Jahr Praxis-Erfahrung gesammelt hatte) was Softwareentwicklung (-reengineering) "im Grossen" und im produktiven Einsatz für Probleme mit sich bringen kann. Neu für mich waren die Methode und die verschiedenen Techniken, mit welchen solchen Problemen begegnet werden kann. Dieser Kurs hat sehr schön gezeigt, dass der "Top-Down-Approach" und die "schrittweise Verfeinerung" wohl sehr nützliche Techniken sind, jedoch in grossen Teams und komplexen Systemen ein strukturiertes Vor-

gehen wie das SIPV sehr hilfreich und erfolgversprechend sein kann.
Um diese Erfahrung reicher, ging's in der zweiten Woche dann richtig los. In den folgenden Monaten beschäftigte ich mich fast ausschliesslich mit dem CASE-Tool *Maestro II* von Softlab. Da unter dem Ausdruck CASE von "besseren" Zeichnungsprogrammen bis zu Code-Generatoren alles "verkauft" wird, muss ich das erwähnte Produkt wohl etwas präzisieren:

Maestro II ist eine sehr "offene" Software. Sie kann vom Anwender stark auf die eigenen Bedürfnisse zugeschnitten werden. Die gelieferten Standard-Techniken (im Fall Swissair ist das SSADM [Datenflussdiagramme; Datenmodelle; Erfassen von Anforderungen, Lösungsvarianten, Systemvarianten; Entity-Life-History-Diagramme; ...]) können in ein Vorgehen eingebettet werden. Im Fall des SIPV sah das etwa so aus: Schritt X kommt vor Schritt Y; Schritt Z braucht diese Inputdaten und liefert diese Dokumente und Resultate; Schritt U darf erst begonnen werden, wenn die Datenflussdiagramme keine Inkonsistenzen mehr aufweisen; usw. Für diese Vorgehensdefinition wird eine einfache Syntax zur Verfügung gestellt und die so erstellte Methode kann dann in die Projekte kompiliert werden.
Auch die Techniken und die Editoren können vom Benutzer definiert, programmiert und geändert werden. Das dazu nötige Know-How wird vom Vertreiber zur Verfügung gestellt. Obwohl dem Anwender die einzelnen Anpassungsmöglichkeiten auf recht einfache Weise dargestellt werden, ist der Aufwand dafür doch recht gross. Ein so stark konfigurierbares Tool lohnt sich wohl nur für Firmen mit einer grossen Softwareentwicklungsabteilung.
Maestro II unterstützt das ganze Spektrum von der Bedarfsanalyse bis zur Implementierung und dem Reengineering. Meine Tätigkeit beschränkte sich auf die Phasen Analyse und Design. In diesen Phasen implementierte ich das SIPV und testete dieses anhand der Fallstudie aus. Nachdem die zwei Pilotprojekte bewilligt worden waren, kümmerte ich mich um die Übernahme der schon vorhandenen Entwicklungsdaten dieser Projekte, die Installation der Software, die Ausbildung der beteiligten Mitarbeiter und mögliche Software-Wartungskonzepte im produktiven Betrieb.
Mein Praktikum endete leider gerade mit dem Start der Pilotprojekte.

Meine Erfahrungen
Ich habe mein Praktikum sehr positiv in Erinnerung. Erstens weil ich nicht dauernd allein vor dem Bildschirm sass und Algorithmen produzierte, sondern mit Leuten diskutieren und mich über Abläufe informieren musste. Zweitens weil das Praktikum ein Gebiet behandelt hat, welches in unserem Studium sicher zu kurz kommt und ich deshalb viel dazulernen konnte. Drittens weil ich das Gefühl

habe, in diesen 14 Wochen etwas geleistet zu haben, von dem andere Leute profitieren und das sie wiederverwenden können. Und "last but not least" weil das Umfeld im Team gestimmt hat, und die Betreuung trotz zeitlichen Engpässen und Überlastung der einzelnen Personen hervorragend war.
Als meine wichtigste Erfahrung möchte ich zwei Dinge erwähnen: Eine Methode wie das SIPV bringt nur dann etwas, wenn die beteiligten Projektmitarbeiter zu einem strukturierten Vorgehen bereit und zu "abstraktem" Betrachten von Problemen fähig sind – dann dafür aber viel.

Die zweite wichtige Erfahrung ist, dass wir (ich!) für die Übergabe des grossen Know-How's, das ich während den dreieinhalb Monaten sammelte, (viel) zu wenig Zeit eingeplant hatten.
Abschliessend kann ich jedem IIIC-Studenten ein Praktikum in diese Richtung und auch ein Praktikum bei der Swissair empfehlen (bei guter Leistung geht's vielleicht wie bei mir: sehr günstig ab in die Lüfte ...)!

Hansjörg Buchser, IIIC/7

Musterlösungen für die Vordiplome der Herbstsession 93

Alle halben Jahre wieder kommt die Bitte an die fleissigen Leute, die in der letzten Session ein Vordiplom absolviert haben:

- Du hast doch sicher in dem einen oder anderen Fach relativ gut abgeschnitten.
- Du hast Dich doch sicher auch gefreut, dass Du beim Vorbereiten auf das Vordiplom Musterlösungen zur Hand gehabt hast. (Na, ahnst Du, worum ich Dich bitten möchte?)

Ja, wie Du (vielleicht) weisst, machen sich diese Musterlösungen nicht von selbst, und bei mir ist das schon viel zu lange her, als dass ich sie machen könnte. Also möchte ich gerade Dich bitten, eines dieser begehrten Papers in unserer VIS-Vordiplomssammlung zu publizieren.

Für das 1. Vordiplom gibt es genug Musterlöser (ja, das ist sehr lieb, dass sich da so viele gemeldet haben).
Aber: **BEIM 2. VORDIPLOM SIEHT ES SEHR LEER AUS!** Ich habe hier in keinem einzigen Fach eine Zusage für eine Musterlösung. Ihr lieben 5. Semester-StudentInnen, die Ihr den Sprung ins Fachstudium geschafft habt, fasst Euch doch ein Herz und verhelft uns zu ein paar Musterlösungen (bittebittebitte)!
Nicht zu vergessen:
Wer ein Vordiplom löst, hat sich ein Abendessen in einem fidelen Kreis mit dem VIS-Vorstand verdient. Hat Dich das vielleicht doch noch überzeugt? --- Hehehehe!

lj

Antritts- und Abschiedsvorlesungen

Es folgt der zweite Teil meiner persönlichen Antritts- und Abschiedsvorlesungspräferenzen. Diesmal ohne viel Worte gleich zur Sache:

Prof. H. T. de Hahn, "etc." (Tip: Es geht um Architektur, nicht um UNIX, pal), 13. Jan. 1994

PD Dr. M. Reiser, "Leistungstheorie – eine noch junge Ingenieurwissenschaft", 31. Jan. 1994

Prof. Dr. W. Guggenbühl, "Elektronik – vom Funk zur Schlüsseltechnologie", 1. Feb. 1994

Prof. Dr. T. Kopp, "Speicherungsmöglichkeiten für regenerative Energie", 2. Feb. 1994

Prof. Dr. H. J. Matthias, "Metrologie, eine ebenso propädeutische wie intergrierende Wissenschaft" (S-Bahn, nicht Wetter, wenn ich nicht irre, pal), 4. Feb. 1994

Prof. Dr. R. Schmidt, "Zur ökonomischen Diskriminierung von Frauen – Begriff, Ausmass, Konsequenzen", 16. Feb. 1994

Prof. Dr. K. Bättig, "'Drogen' psychologisch gesehen", 18. Feb. 1994

Alle genannten Veranstaltungen finden an den genannten Tagen um 17.15 Uhr im Auditorium Maximum statt (HG F-Stock).

pal

Gödel Numbers: A new Approach to Structured Programming

Norman H. Cohen
Software Research Group
Sperry Univac
P.O. Box 500
Blue Bell, PA 19424
USA

Views expressed herein are those of the author exclusively, and not those of Sperry Corporation and its Sperry Univac Division. No implication is intended as to any product commitment.

One of the earliest and most fundamental ideas in the history of computation was the notion of Gödel numbers. Turing [2], after asserting that every computable function corresponds to a computing machine (of the kind now known as a Turing machine), showed that every such machine could be mapped into a unique natural number. As a consequence, the computable functions are enumerable: There is a correspondence between the natural numbers and the computable functions, such that each computable function corresponds to infinitely many natural numbers and each natural number corresponds to a unique computable function. Because Turing's scheme for assigning a number to each computing machine is reminiscent of a scheme by Gödel for assigning a number to each wellformed logical formula, the numbers corresponding to computable functions are known as Gödel numbers: the correspondence itself is often called a Gödelization.

In the spirit of [5], we propose the application of a concept heretofore monopolized by theorists to the solution of practical programming problems. Specifically, we call for the elimination of conventional programming languages in favor of "programing" by Gödel numbers. Instead of writing a computer program in a language like Pascal in order to solve a problem, a "programmer" need only feed to a computer the Gödel number of the computable function which solves his or her problem. The computer, playing the role of a "universal Turing machine," would then solve the problem.

Example: In order to compute factorials, it would only be necessary to specify the Gödel number

5922623454066207111311435351265955
1651504629.

(It is important to emphasize that we are not proposing computations actually be performed by simulation of Turing machines. We are suggesting that Gödel numbers be used to specify the function to be computed, not the exact method for computing the function. A sufficiently clever optimizing compiler can translate a Gödel

number n into efficient machine code for the n-th computable function.)

One of the most important principles of structured programming is what Dijkstra [7] calls "separation of concerns": When thinking about a program's input-output relation, one should not be concerned with the internal mechanisms of the program. Programming by Gödel number is an excellent way to enforce this principle. There is no notion of internal mechanisms, since a program is the number identifying its input-output relation.

The other obvious contribution of Gödel numbers to structured programming is that it enforces goto-less programming. The dangers of programming with goto statements are eloquently described by Dijkstra [11]. Since Gödel number programs consist entirely of digits, the presence of a goto will be detected quite readily during the lexical analysis phase of compilation. (An alphanumeric languages without goto statements, the characters GOTO may still appear to be a valid identifier. This means that the presence of a goto statement cannot be detected until the syntactic analysis phase of compilation.)

In general, programming with Gödel numbers reduces the programmer's need to be concerned with details of syntax. In fact, on a keypunch or terminal which only accepts numerical input, it is impossible to produce a syntactically incorrect program. This should lead to higher productivity, since a programmer can avoid the two or three initial runs which remove syntax errors from his program, and proceed immediately to test the runtime behavior of his or her Gödel number.

Gödel number programs are a boon to the software engineer. Gödel numbers are eminently portable, since they run on any universal Turing machine. Version control and library maintenance become trivial, since Gödel numbers serve as indices uniquely identifying themselves. There is no longer any danger of pitfalls such as two programmers using the same name for different programs: A Gödel number "program" is its name.

Before Gödel numbers can be put to practical use, it will be necessary to establish a standard Gödelization.

Currently, every recursive function theorist has his or her own favorite Gödelization scheme. Gödel [3] describes a Gödelization based on the Unique Factorization Theorem. Turing [2] encodes descriptions of computing machines using an alphabet of seven symbols, and then translates each symbol into one of the digits 1, 2, 3, 4, 5, 6, 7, to obtain a "description number" for the machine. Kleene [13] uses a similar approach, based on Turing machine encodings over an alphabet of fifteen symbols, which are then translated into pentadecimal

numbers. In certain contexts, Kleene "closes the gaps" in the Gödelization by then enumerating those pentadecimal numbers which correspond to valid Turing machine encodings, and using the position of a machine's encoding in this enumeration as a Gödel number. Hopcroft and Ullman [17] use five symbols to encode Turing machines, define all ill-formed strings to be encodings of a Turing machine with no moves, and use the position of the encoding in an alphabetic enumeration of all strings over those five symbols as a Gödel number. Rogers [19] has the temerity to write, "We assume now that we have selected one such listing procedure. We keep it fixed for the remainder of the book. We do not give formal details." Perhaps Rogers can be forgiven: There are those who would not even keep the listing procedure fixed for the remainder of the book!

If Gödel numbers are to be used effectively by practitioners, it is crucial that we put an end to the theoreticians' handwaving. The American National Standards Institute must form a committee in the very near future to develop a standard Gödelization. This will not be an easy task: A practical Gödelization should have relatively small Gödel numbers for the most commonly computed functions, such as payroll, accounts payable, Gaussian elimination and space war. The standard Gödelization should be adopted by the National Bureau of Standards and the United States government should not procure any Gödel numbers which are not based on the American National Standard Gödelization. Without a strictly enforced standard, Gödel numbers will go the way of quadraphonic.

It would be irresponsible to propose programming by Gödel number without considering the social consequences of this choice. Perhaps the most profound social effect would be that programmers could be replaced by monkeys sitting at terminals. Every Gödel number entered by a monkey represents some computable function. This function may not be precisely the one called for in the specifications, but at least the monkey would have something up and running very quickly. This is comparable to the performance currently obtained from many human programmers. Another social consequence, alluded to earlier, is that the terminals at which the monkeys sit will be cheaper: There will be no need for non-numeric input.

Teachers of recursive function theory will enjoy increased prestige: Not only will their ideas have found practical use: their constructions and proofs will seem more concrete, and thus more believable. Instead of saying "Assume some suitable Gödelization," one could say "Assume the American National Standard Gödelization." Instead of saying "Let k be the Gödel number of the universal Turing ma-

chine," one could name that specific value.

Lawyers will enjoy increased employment, as courts determine the implications of Gödel number programming for proprietary software. Mooers [23] has pointed out that although software cannot be patented, it can be protected by copyright. But can one copyright a number? If so, may the phone company print in its directory a phone number which also happens to be a copyrighted Gödel number? If we adopt the view of Principia Mathematica [29] identifying a cardinal number n with the set of all sets of n elements, then do proprietary rights over a Gödel number n imply proprietary rights over that set? Do proprietary rights over a set extend to its elements?

An attractive refinement to programming by Gödel number is programming by unary Gödel number. That is, the Gödel number k can be represented by a string of k occurrences of the digit 1. The resulting exponential growth in the length of Gödel numbers is an admitted drawback. But many of the advantages of programming by Gödel number would be amplified: Terminals could be made even cheaper, since only one printable character will be required. Programming monkeys could be replaced by programming chicken.

The universal Turing machine is one of the first computing devices ever designed. At last the time has come to harness the power of this design to fulfill the dream of error-free programming. When programming with Gödel numbers, there is no such thing as an incorrect program. Every number is a correct program, though perhaps for the wrong task.

Submitted by E. Birrer

Literaturliste in [59], pal

ASCII-COWS

```
                /\   (*
               /__\  |
               (oo)  |
         /-------&&  |
        / |      |&&  |
      *   ||----|\___|
          ^^     ^     |
```

Santa Cow

Berichtigung!

In der letzten Ausgabe der Visionen ist mir ein bedauerlicher Irrtum passiert: Die Durchschnitte in den Fächern des 1. Schlussdiploms stimmen überhaupt nicht. Sie stellen das Resultat einer unvollständigen Copy-And-Paste-Aktion dar. Deshalb werden hier die Fachdurchschnitte (diesmal 2x von jedem Vorstandsmitglied nachgeprüft) richtig abgedruckt:

	Schnitt	Varianz
Rekurs. & Kompl.	4.74	0.36
Digitalt. & Rechnerstr.	4.66	0.56
Informationssysteme	4.69	0.41
Systemsoftware	4.63	0.76

Im Text wurde auf die falschen Durchschnitte bezug genommen. Wie man sieht, hat kein einziges Fach einen Durchschnitt unter 4.4. Alle Fächer haben sogar eine Durchschnittsnote von über 4.60, was sehr positiv hervorzuheben ist.

Die anderen Daten aber, so möchte ich betonen, sind absolut korrekt!

Es tut mir sehr leid, dass dieser Fehler passiert ist, ich möchte mich dafür entschuldigen.

lj

Berichtigung ^{2}!

Zum Artikel "Programmschutz durch das neue Urheberrecht" in der letzten Visionen-Ausgabe: Durch ein Missgeschick gingen bei der Text-Konvertierung zahlreiche Gänsefüsschen, Apostrophe und ein Gedankenstrich verloren, ausserdem wurde der letzte Satz abgehackt. Die Redaktion bittet für allfällige Unklarheiten um Verständnis. Die Korrekturen im einzelnen:

- S. 18 mitte rechts: ..."wörtlich"...
- S. 19 mitte links: ..."Abweichen vom Vorbestehenden und vom zu Erwartenden"...
- S. 19 mitte links: ..."statistische Einmaligkeit"...
- S. 20 rechts: ...PB's ...PA's...
- S. 21 mitte links: ..."Nachmachen"...
- S. 21 unten links: ..."fremden"...
- S. 21 oben rechts: ..."ausbügeln"...
- S. 21 oben rechts: ..."Gelegenheits"-Programmen...
- S. 21 unten rechts: ...konzipiert – entsprechend ...
- S. 22 oben links: ..."dirty"... "sauberer"...
- S. 22 mitte links: ..."Abweichen vom Vorbestehenden und vom zu Erwartenden" ...
- S. 22 oben rechts: ..."nie" ..."im Wege"...

- Der letzte Satz lautet komplett: Der Gesetzestext zum neuen URG ist bei der EDMZ in Bern erhältlich (Nr. 33.314d, Fr. 2.30 gegen Legikopie).

pal / C. Reuss

Opinions about C and C++

Ellis MA, Stroustrup B: ... the C array concept is weak and beyond repair.

Sakkinen M: The C logic, which C++ has been more or less bound to follow, seems to have gone like this: (1) All arguments shall be passed by value; that is clean and simple. (2) We cannot afford to pass arrays by value; that is much too inefficient. (3) Let us invent a trick to reconcile (1) and (2): the name of an array shall not denote the array but only the address of its first element.

Meyer B: The search for compatibility at any cost is also the reason behind the centaurs sporting an object-oriented head on top of a C body, such as C++. Imagine this: on the one hand, inheritance, on the other hand, pointer arithmetic!

Sakkinen M: Many seminars and tutorials even on topics like object-oriented design have the clause with C++ in their names or advertisements to assure potential attendees of staying safely in familiar territory.

Edelson D, Pohl I: C, however, is also transparent; C++ is not.

Sakkinen M: Since C supports both structured programming and strong typing only halfway, we might say that the kind of halfway object orientation that C++ offers suits the style well.

Edelson D, Pohl I: C is an elegant language; it is small and simple. The syntax of C++ is similar to that of C, but its semantics are neither small nor simple.

Sakkinen M: Incrementing C by 1 is not enough to make a good object-oriented language.

Übermittelt von
Manuel Bleichenbacher, IIIC/7

Beruhigungs-Rechteck

Sollten Sie sich unsicher, unwohl oder beunruhigt fühlen, drücken Sie bitte auf dieses Rechteck:

```
---------------
|  KOPF HOCH  |
---------------
```

In Momenten grosser Anspannung ist es oft sehr beruhigend, physischen Kontakt mit vertrauten Gegenständen herzustellen.

Dieses Rechteck ist Ihr Freund.

Aus alt.de.instant.remedies

F&K's Bieridee

Wie schon in den letzten Visionen versprochen, möchte ich Euch heute noch etwas detailierter in die Geheimnisse des Bieres einführen. Doch auch hier gilt: Kein Lehrstoff ohne Übungen.
Das heisst: Bevor Ihr weiterlest, benötigt ihr:
- 1 Flasche Pilsener (etwa das *Pilsener Urquell*)
- 2 Flasche Lager Hell (*Calanda Bräu*!)
- 2 Flasche Münchener Starkbier, dessen Name auf "ator" endet, also z.B. *Salvator* oder *Optimator*
- 2 Flasche Altbier (sehr zu empfehlen ist das *Diebels Alt*)
- 2 Flasche Weizen (Natürlich kommt hier von den Schweizer Bieren, die in Flaschen erhältlich sind, nur das *Calanda Weizen* in Frage oder sonst ein feines Münchener wie etwa *Schneider Weisse*)
- ein Weizenglas, drei normale Biergläser und ein Pilsenerglas, alle mit kaltem Wasser ausgespült.

So, nun könnt Ihr weiterlesen. Aber lasst die Flaschen (noch) zu!

Obergärig / Untergärig

Diesen Begriff hört man immer wieder im Zusammenhang mit Bier, aber nur die wenigsten Leute, die ich einmal darauf ansprach, konnten korrekt Auskunft über den Unterschied geben. Deshalb das ganze nochmal: Der ganze Unterschied liegt in der Hefe, einem der Hilfsstoffe, die man braucht, um Bier herzustellen. Obergäriges Bier wird mit obergäriger Hefe gebraut. Analog: Untergärig. Untergärige Hefe braucht eine kalte Umgebung von etwa 4 bis 9 Grad, obergärige mag die Wärme, so um die 15 bis 20 Grad. Die obergärige Hefe heisst so, weil sie während des Gärvorganges nach oben steigt; untergärige Hefe sinkt auf den Gefässboden ab. In den Anfängen der Braukunst in Europa gab es nur obergärige Hefe. Dies stellte die Klöster (Brauereien...) vor riesige Probleme: es konnte nur im Winter gebraut werden, da im Sommer die Hefe verdarb. Da Bier aber auch im Sommer ein Genuss ist (you know) wurden immer tiefere Höhlen gesucht, in denen Bier gelagert und zum Teil auch hergestellt werden konnte. Ausserdem wurde die untergärige Hefe entwickelt, mit der man auch im Sommer brauen konnte. Somit war die Versorgung mit dem Lebenssaft das ganze Jahr gesichert. Dann wurde auch noch der Kühlschrank erfunden (komischerweise nicht von Brauereien), was alles noch viel einfacher machte.
Doch wie sieht der Unterschied der Biere vor Dir auf dem Tisch aus? Also, sortiere die Flaschen nun mit dem ober/untergärig-Algorithmus. Das Resultat sollte die Menge der Biere in zwei Teilmengen O und U unterteilt haben. Dabei ist U = {Pilsener, Calanda Lager, Münchener Starkbier} und O = {Altbier, Calanda Weizen}. Halt! Noch nicht trinken! Das war erst der Teil über ober- und untergäriges Bier...

Die U-Biere
Nehmt nun das Pilsenerglas und zwei normale Biergläser und stellt diese vor Euch auf. Schenkt nun das Pilsener fachgerecht mit viel festem Schaum ein. Danach das Calanda Lager und das Münchener Starkbier. Dieser Vorgang sollte in etwa acht Minuten abgeschlossen sein. Nun wird in dieser Reihenfolge probiert: Pilsener, Calanda Lager und Münchener Starkbier. Wie Ihr merkt, nimmt die Bitterkeit bewusst ab, der Alkoholgehalt des Bieres (und Eures Blutes) zu. Wie gesagt, alles untergärige Biere, bei denen die Stammwürze massiv zugenommen hat. Dies ist auch das Stichwort für die nächste Bieridee: Die Stammwürze. Prost1!

Die O-Biere
Nun werden die Gläser auf die Seite gestellt (nicht abwaschen!). Nehmt nun das Weissbierglas und ein normales Bierglas. Schenkt das Weissbier (Weizen) ein, achtet aber darauf, dass dieses eventuell sehr viel Schaum entwickelt (!) und schenkt dann auch das Altbier ein. Probiert nun als erstes das Alt. Es ist leicht bitter, aber bei weitem nicht so bitter wie ein Pilsener. Typisch ist, das dieses Bier wenig Kohlensäure enthält, ganz im Gegensatz zu dem jetzt endlich zu probierenden Weiss-/Weizenbier. Hier ist die Kohlensäure ein wesentlicher Bestandteil. Der Name "Weizenbier" kommt übrigens daher, dass die Gerste zu einem mehr oder weniger grossen Teil durch Weizen ersetzt wurde. Prost2!

Der O/U-Test I
Falls Ihr den Test bis hierher an einem Abend gemacht habt: gratuliere! Aber jetzt wird es anspruchsvoll: Den Geschmack von O- und U-Bier direkt zu vergleichen. Nehmt nun die Gläser, die Ihr für das Calanda Lager und für das Altbier benutzt hattet, und schenkt Euch die zweite Flasche des jeweiligen Bieres ein. Jetzt habt Ihr die einmalige Gelegenheit, den Unterschied von O und U selbst zu erleben! Prost3!

Der O/U-Test II
Dieser Test Läuft analog zum O/U-Test I ab. Nur dass Ihr hier das Münchener Starkbier mit einem Weizen/Weissbier vergleicht. Dieser Test ist für Liebhaber von sehr vollmundigem Bier ein Fest. Er gehört übrigens zu meinen Lieblingstests und nun könnt Ihr wohl auch verstehen warum. Prost4!

In den nächsten Visionen: Die Stammwürze

Frank

P.S. Der Autor lehnt jede Haftung für physische, psychische und sonstige Folgeschäden ab. Er hat den Test aber selber durchgearbeitet und erfreut sich heute noch des Bieres!

G.A.B. 6648 Minusio

pal

Falls unzustellbar bitte zurück an:

Verein der Informatikstudierenden
IFW B29
ETH–Zentrum

CH–8092 Zürich

Inhalt

Adressen	S. 2
Hei Folkens!	S. 3
Redaktionsschlüsse 1994	S. 4
10 Jahre VIS	S. 7
Zehn Studenten in Swansea	S. 9
Rangliste in Swansea	S. 14
Chris Flu's Feldkochecke	S. 16
Büchertip: Unternehmen Zufall	S. 19
Lerngruppen	S. 20
C-Kurs, Teil 1	S. 22
Abteilungsnews	S. 33
Praktikum bei der SWISSAIR	S. 36
Programming by Gödel	S. 40
F&K's Bieridee	S. 46