EUUG

EUROPEAN UNIX® SYSTEMS USER GROUP NEWSLETTER

INDRE BY-TERMINALEN
ved Københavns Universitet
Studiestræde 6 over gården
DK-1455 København K
Telefon 01 - 12 01 15

Volume 3, No. 4
WINTER 1983

EUUG

European UNIX† Systems User Group

Newsletter Vol 3 No 4
Winter 1983

† UNIX is a Trademark of Bell Laboratories.

Monday 16th of April to Wednesday 18th of April 1984

EUUG Spring CONFERENCE

Nijmegen, The Netherlands

If you want to know more about UNIX, if you really are a part of the UNIX world... one place you should go before you take off for your Easter holidays in April is the **EUUG conference** in **Nijmegen, The Netherlands**.

Follow the informative technical sessions about the current UNIX developments in graphics, text-processing, networking, distributed UNIX systems, programming environments, advanced editing, database management, etc...
or follow the tutorials in UNIX: advanced editing, advanced (c)shell programming, advanced textprocessing, the programming language C and programming style, UNIX UUCP networking, UNIX 4.2 BSD networking, UNIX machines and market situation, licensing and System V.2,
or take part in the panel discussions or *birds-of-a-feather* sessions.

For the conference the following famous speakers are invited:
Brian Kernighan (Bell Labs, Murray Hill, USA)
Brian Redman (Bell Telephone Labs, USA)
Eric Allman (Britton-Lee, USA)
These speakers do not need further explanations about what they have done.

Proceedings of the technical talks will be available after the conference.

For attendees of the technical sessions and tutorials the lunches on Monday, Tuesday and Wednesday are included in the attendance fee.
During the conference an **exhibition** with all the major UNIX vendors will be open from 9 am till 6 pm.
On Monday evening a welcome **buffet** will be organised in one of the hotels.
On Tuesday evening a **dinner** is offered. Because of limited space you are advised to have the dinner reserved. Only one extra reservation for one companion is allowed.

The conference is the only UNIX event to offer you the most complete information on UNIX and the only real opportunity to meet UNIX people from different countries.

Pre-registration for attendees and hotel arrangements should be directed to the EUUG secretary. Tax is included in the fees this time.

fee	**techn. sessions**		**tutorials**		**exhibition only**	
registration	*members*	*non members*	*members*	*non members*	*members*	*non members*
Preregistration	£130	£160	£130	£160	£5	£10
on site registration	£160	£200	£160	£200	£5	£10

Hotels: You are advised to make hotel reservations before 12th of March! The EUUG has made hotel arrangements at special cheap rates. Reservations (different qualities and prices) can be made via Tourist Inf. Office VVV, mrs E. v.d. Ven, St. Jorisstraat 72, 6511 TD Nijmegen, Holland, tel: +31 80 225440 or telex 48228 winat nl attn H.J. Thomassen. A deposit of Dfl 150.- per room is needed. Make sure the office knows that you made the arrangements because of this EUUG conference.
Sharing of rooms should be arranged via the hotel directly. The hotel prices for conference attendees varies from Dfl 67.50 to Dfl 150.00 (breakfast and taxes included). A special hotel shuttle bus from and to the conference hall has been arranged.

Some national groups will organise special flights, registration and hotel arrangements. Please contact the secretary of your national group for information.

Location: Conference office, University of Nijmegen, Faculty of Computer Science, Toernooiveld, 6525 ED Nijmegen, The Netherlands.

Exhibition booth reservations: For booth reservations please contact the EUUG secretary or the local organiser: H.J. Thomassen, Faculty of Comp. Science, Toernooiveld, 6525 ED Nijmegen, The Netherlands, phone +31 80 558833, telex 48288 winat nl.

After registration has been made with the EUUG, you will receive a conference information package with agenda, hotel arrangements, location information, etc. At the registration desk you will receive the name badge, tickets for lunches, dinner, etc.

Sight seeing: For Thursday, the 19th of April, the EUUG will try to arrange a special trip to the **Delta Works** (water works in the south of Holland). This arrangement is dependent to the number of subscribers (at least 40). The cost for this trip will be about £30.

see YOU in NIJMEGEN

UniForum January 1984
Visiting the Zoo in Washington

Teus Hagen

Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands

The Zoo

During the week of the 17th of January 1984 I visited the zoo in Washington. Everyone at my institute thought I was attending a conference in Washington. Well, they were wrong. Instead I visited the zoo in the basement of the Washington Hilton Hotel. According to the figures of the organiser of the conference (/usr/group, USENIX was co-sponsoring it), the floor was crowded with about 9000 visitors. Well, I had three badges, so there were at least 8997 other 'Homo Sapiens' around. The zoo also exhibited some elephants (like Pyramid, Gould, and DEC VAXes). Some of the elephants just showed their new offspring, like the MicroVAX. Other cages had the giraffes (NCR tower, Cadmus, Zilog and Gould families), lions (Altos, Zilog and Pixel), work-horses (SUN, Dual, Ridge) and even the little mouses were on show (IBM PC/IX, Tandy, Apple and Gould again). In total, 150 different cages in the zoo to bring joy into the life of a simple programmer. Pity, after one and a half days the zoo was closed, the joy was over for Bill.

The exhibition

Some aspects of the exhibition were worthwhile to visit. Choice enough, for nearly all the old UNIX families (hardware and software) presented their goodies. One of the newcomers to the exhibition floor was: AT&T. They showed their Blit (Robert Pike), called the Teletype 5620 and some of the AT&T workbenches.

Amazing was what one can do in a year to have a better quality of UNIX running on a little machine. Let's say experience showed that it cost one year to really have UNIX running after the port is done.

All the big companies had small booths, IBM with the PC/IX machine (based on the Intel 8088 chip) of which the rumour is that the software price will be around $900!, the three chip manufacturers, Motorola (two types of machine), National Semiconductor (Genix) and Intel (who allowed me this time to run some benchmarks), and the old guys like Amdahl (really fast), Hewlett Packard (neat hardware), NCR (the 'Tower'), Honeywell (two types of machine) and Texas Instruments (a newcomer with TI NU). And DEC, who were really the first in the UNIX world, presented their MicroVAX and talking chip. Again no sign of BBN at the exhibition, have they left the UNIX battle field?

The real news was coming from some new manufacturers like Pyramid, Ridge and Silicon Graphics (Steve Bourne is there now). The last showed their space-shuttle flight simulator (3D colour and real time!). That machine filled my eyes with dollar notes. The main trend is to machines based on the Motorola chip, with lots of cache space (4K bytes at least), colour rasters, lots of memory (like 2Mb), and of course super fast (more than 1 MIP).

In general, the market can be divided into four categories of machine:

- A display (white, green, yellow or blue) with a built-in 10Mb Winchester and a floppy. Most do not have real LAN hardware or software capabilities. Personally, I wouldn't like to have one of those awful looking monsters on my desk, even if some of the monsters look like a stack of books (Gould). Most have a dialler, modem and uucp possibilities; these WAN capabilities will be discussed later.
- To try to get away from the awful packaging, the tower-profile was invented (by DEC or NCR?). Almost all manufacturers produce machines in this form (SUN, Pixel, Zilog, Cadmus, etc). Only because of the noise of the fan you will notice the machine under your desk. I could not find the TI NU machine until I saw an advertisement for a low noise fan from Texas Instruments, one does not expect that from someone from Texas.

- Real intelligence can be found in the 'workstations'. Many of them are sold with a raster display. Some examples are the SUN and Cadmus (PCS in Europe) machines.
- The old computers, which you should hide in a computer room: big boxes like VAX (DEC), Pyramid, Arete and the fast Ridge machine. The Amdahl machine was really hidden somewhere.

Some remarks about the (new) machines:

MicroVAX from DEC: it looks like a Micro-11 and behaves like a child bred from a VAX730 mother and VAX750 father. The child walks away with some short Q-bus based legs.

The new Pixel P/35 breed runs just fine. Like Altos and Cadmus they have a LAN implementation based on the Newcastle ideas.

The Texas Instruments NU machine is nice and quiet. They did some good work on the fans and have a new bus architecture (from MIT).

The Pyramid machine was facing the world with 128 registers (or was it 256) in mind. Another fast brother was the Ridge machine, advertised with a speed of 8 MIPS. It was so fast that I good only discover 4 of its MIPS. Anyway, fast enough for me.

The National Semiconductor 16032 chip runs at 6 MHz now, so the IBM PC/IX machine can now run on the 8088, 68000 and 16032 (both from Sritek). One IBM PC was running with a Fujitsu Eagle (440 Mb) at the Sritek booth.

The new Cadmus firm sure know how to advertise, their black box looked like it came from John Player, and it looks like all you can get out of it is cigarettes. I prefer the same machine in a different shaped box and with software that is really running, perhaps they should ask PCS in Germany how to do it.

On some of the machines a little known benchmark program was run. Due to some requests the results will be reprinted in this newsletter.

Software

AT&T, Unisoft, Mark Williams, Microsoft, Wollongong, Whitesmith and HCR are trying to take the lead in software. Certainly it is not longer true that there is no software available under UNIX, see the new edition of the '/usr/group' catalogue to get an impression, but also look at the small interesting ones: Sphinx, Unify, and ...

Give me a name and I'll give you a database management package. All kinds of editors, like the interesting 'Interleave' editor, were presented, and Emacs in all shapes, colours and flavours. Network software as if nobody was communicating at all.

Journals and Other Paper Work

Yes, there can't be a fashion without a journal totally devoted to it. Two new journals totally devoted to UNIX: UNIX Review and UNIX/World. In the latter journal I noticed the best articles. Both carry the same advertisements you find in in Byte, Electronics, Mini Systems, etc. None of the two is pro deo.

Yates were presenting their market overviews.

The Decease of the Zoo

What do you bring to home from a zoo? Well, besides some strange disease I collected the following items: collection of T-shirts (thanks to NIXU and SOL); muppets (Computer Automation, Unisoft); the 'grep' milk cup (Amdahl); disco cap, screw driver and poster (DEC); screw driver (Sritek); yellow matchbox bus (TI NU); plastified business card (Yates); USENET map (plotted) (HP); the name badge of Bill Joy (Usenix); the name badge of Otis Wilson and Bill Murphy (AT&T); and I'm still waiting for the free IBM PC/IX machine (IBM). Others had nothing but joy for me.

The Dinner Problem Solved

After one day of walking through an exhibition you get hungry. Well that didn't turn out to be a problem. Every 'good' company organises a hospitality suite, and you get invited with an invitation card, but even if you don't get a card, it is still no problem. Every floor in the Hilton Hotel had about three hospitality suites, and as there are only ten floors available in that hotel, it was impossible to leave the hotel without being full of cheese, fruit and liquid. The only problem arose on Friday night, no hospitality suites left.

Yes, AT&T arranged, of course, the biggest 'brunch' (or better lunner) than ever.

Technical Talks

Due to a fully filled agenda with meetings, I had no chance to get a good overview of the talks, some of which promised to be quite interesting.

Some of the panel sessions were guided by managers and (technical) directors of companies. Well, I discovered now that marketing UNIX or talking about the future aspects of UNIX can be made really boring. And surprisingly, the technical panel sessions tend to be the same exercise. It is becoming difficult now to attend some real interesting talks.

So some real new announcements from AT&T: the document workbench (another, better, ditroff and lots of document processing stuff), BASIC interpreter, and the MC 68000 software generation system.

The new System V.2 release is out now. It has a few new programs as f.i. mailx, pg, qasurvey and trenter. Lots of improvements like f.i. job control, F77 compiler, shell and new flags for the 'ls'-command. A better documentation set and lots of bugs solved again.

System V will have a library with 'high quality' application programs (editor: third party software) and software for several different computers. The package will be made available in co-operation with a newcomer to the UNIX world, **Digital Research Inc.**

For re-sellers of the System V software there are more flexible royalty schedules. AT&T is talking now to the different OEM's to make new arrangements.

And if we call the initial System V release V.0, and have now V.2 announced, what can we expect if AT&T has internally V.4 and V.6 running? Is V.8 the next one? Or is it just Edition 8 from Bell Labs?

The UNIX Network is Changing

Most of an afternoon session was dedicated to networking. UUCP has been re-written, most of the speed-ups are about the same as is distributed with the EUUG D2R3 software. That is: all data about a site is moved to a special 'site' sub-directory, the error messages make more sense, the login-protocol handler is table driven, the table has entries for all kinds of modems, diallers, port-selectors and internal line configurations. Special care has been taken as regards security, per directory one can specify read, no-read, write, or no-write permissions. Also, the specifications for a site can be made more easily. It works in the same way as the 'termcap' file: f.i.

```
logname = uucpa \
machine = you \ # we call you
validate = raven \ #if you say you are "raven", check for logname "uucpa"
commands = rmail:rnews:uucp \
write = /usr/spool/public:/usr/spool/news \
nowrite = /usr \
noread = /etc \
sendfiles = yes
```

Some capabilities have been added: public encryption to identify remote sites, automatic internal data storage/transmission encryption and gateway facilities.

What is still missing was 'grading', to get mail sent before news is shipped, and 'nicknaming', to

facilitate the change of names.

Mark Horton expects to see a fast growth of the total number of sites connected to the UNIX network. Every PC sold now has a modem, sometimes a dialler, and UUCP. Tandy, IBM and Apple will have a production of some thousands of machines per week, so a figure of 100,000 sites (and names) by the end of 1985 is possible. To try to meet the problems which will come with it, a decision was made to structure the UNIX network in the US in the way it is done in Europe. Within a year one hopes to have created domains of countries, of states or of geographic surroundings, of big companies (AT&T?) and of course with subdomains. The address scheme will be much simpler: *teus@mcvax.UUCP* instead of the string *cbosgd!philabs!mcvax!haring!teus*. The routing information will be kept in the backbone sites, because the database will simply not fit in a PC (yet). Also, time-costs will be included in the routing scheme. An inquiry form to get all the information will be published shortly. The hope is to have a domain oriented addressing scheme in production by the end of this year.

User Group Arrangements

After the talk of Jim McKie 'Where is Europe' at the Toronto conference, the EUUG got ten minutes in the opening session of /usr/group to explain that 'Europe is Here' (Emrys Jones). The representatives of the EUUG had some discussions with the US users groups and made the following agreements:

USENIX: all services given by a group are available (at cost price) to the members of the other group, so in essence the only thing you don't get is voting rights in the associated group. For members of the EUUG:

- Publications like newsletters, announcements and proceedings from USENIX are available via the EUUG secretary. The EUUG secretary can arrange bulk shipping of that paperwork, f.i. the EUUG secretary already has copies available of the Toronto Conference Proceedings (500 pages per book). The price is $30 per copy, if you order it via USENIX you have to pay $15 extra for overseas postage.
- Software distributions are available via the EUUG Distribution Center. Special arrangements have been made with AT&T to make this exchange of software across the Big Puddle possible. A new distribution is on its way now.
- The attendance fee for USENIX conferences is the same as for USENIX members. All arrangements for those conferences should go via the EUUG secretary.

STUG: The same arrangements were made with the Software Tools Users Group. The EUUG is already distributing the STUG software tapes, and a new tape with the communication software to exchange data between machines with different operating systems will be available soon.

/USR/GROUP: With /usr/group arrangements were made to exchange publications. Final arrangements for bulk shipping of their 1983 catalogue are being made now.

Summary:

- The newsletter **;login:** and the **Toronto Proceedings** can be ordered now from the EUUG secretary ($30 for the Proceedings).
- The USENIX **Software Distribution 84.1** can be ordered from the EUUG Distribution Center, CWI, Kruislaan 413, 1098 SJ Amsterdam, Netherlands. This distribution contains licenced material, so you have to enclose a copy of your licence (preferably SV and 4.2BSD). If necessary, the EUUG will verify the licence agreement with the licensor. The tape is expected in a few months.
- The /usr/group **Catalog Edition 83** can be ordered from the EUUG secretary. The EUUG secretary will try to organise bulk shipping. The price is $25 (exclusive of postage costs). Price to non-members is $50.

In the future, the secretary of each national group should do the ordering. This will save some time delay as those secretaries will (probably) forward the order electronically to the source.

Conclusions of the Visit to the Zoo

The exhibition is going in the direction of the Comdex fair shows, some booths had only salesmen around with no real knowledge of UNIX at all (I wonder if in the future there will be any technicians around, just in case a screwdriver is needed?).

The 'technical' talks were of a low quality. If not organised in a different way, the talks will end up at the same level as at the Comdex fairs.

A one day course about some 'internal aspects of UNIX' (C style and portability, advanced shell programming, advanced editing, UNIX and LANs) for $100 is really cheap. The general feeling was that those courses were good.

I would like a change in the way the conference is organised now in the US. It is only possible once to get so many people on their feet and give them so little for their money.

The Evolution of the Berkeley UNIX Project

Received from CSRG, Berkeley

The distribution of Berkeley UNIX 4.2BSD to licenced installations began on September 30th, 1983, and is proceeding quite smoothly. In the meantime, the Computer Systems Research Group of the University of California at Berkeley has started working on four new research projects, in addition to further tuning and refining the facilities of 4.2BSD. As of August 1st, CSRG has been headed by Mike Karells, who previously worked with the Berkeley PDP-11 distribution. He is working under the guidance of Prof. Domenico Ferrari while Prof. Robert S. Fabry is on sabbatical this year. Pland have been prepared for the next three years in each of the four areas. The projects will be supported by a new three year contract from the Defense Advanced Research Projects Agency.

The first project will study various issues related to the design of mechanisms for distributed access of files and other resources in a network of single-user workstations running under Berkeley UNIX. The issues include the performance effects of the amount and use of local storage present in each workstation, the design of flow control policies to prevent saturation of such remote facilities as file servers, and the tradeoffs between autonomy and transparency in accessing distributed objects.

Determining the cost-performance impact of several decisions to be made when designing a distributed name server is the main goal of the second project. Various lookup algorithms, local caching, and cache verification policies will be investigated. A Berkeley UNIX-based Name Domain server for the DARPA Internet is being developed to run experiments and to provide parameters for the modelling part of the study.

The third project is concerned with evaluating various metrics and policies for automatic load balancing in distributed Berkeley UNIX systems. All the configurations being considered are based on a local-area network, and include single-user workstations as well as multiple-user interactive machines. The goal of this effort is to design and implement a viable load balancing scheme that can be tuned to the characteristics of different environments and host configurations.

The researchers involved in the fourth project are designing a distributed measurement instrument and a distributed program debugger. The two problems are being attacked together because of their common need for remote process control functions. The measurement instrument should, among other capabilities, allow the experimenters to create and sustain an artificial load on a distributed system, to control remote software monitors, to capture interprocess communications, and to keep the clocks of the various hosts on a local-area network in as full synchronisation as possible. The debugger should allow programmers to control the state of their distributed applications, and verify the correctness of their assumptions about synchronisation and event ordering.

The addition to 4.2BSD of sub-network routing, allowing logical and physical networks (or external and internal addressing) to be different, and the revision of the terminal line disciplines to make them more consistent with the network interface, thereby improving remote terminal facilities, are, among the other projects being considered, the most likely to be undertaken in the near future.

At this time, there are no specific plans for future releases of 4BSD; as the system evolves, the additions will be incorporated into a new distribution when appropriate.

Some Self-Reproducing Programs

Theo de Ridder

IHBO de MAERE
Enschede

It is an interesting and educational exercise to write a self-reproducing program for a given language. To exclude trivial solutions to this well known [1] problem the boundary conditions are:

a) The empty or single token program is not accepted.

b) References to external data or to the internal representation of the program are not allowed.

Analysing the problem in general terms makes clear that the program should contain its own source in two different representations, one for interpretation and one for printing. In most languages this can be realised by quoting symbols. However the two representations cross each other and one has to quote (escape) a quote symbol without using itself.

The basic non-functional solution has 3 steps:

1) Assign the complete source of 1), 2) and 3) to some place.

2) Print the assignment 1).

3) Print the print-statements 2) and 3).

Following the general approach we will first give a solution for three fundamental and typical UNIX languages: ed(I), sh(I), and C.

selfcopy.ed

```
a
s/.*/a/p
u
1,$p
s/.*/./p
u
1,$p
Q
.
s/.*/a/p
u
1,$p
s/.*/./p
u
1,$p
Q
```

selfcopy1.sh

```
A='"' B="'"
C='echo "A=$B$A$B B=$A$B$A"; echo "C=$B$C$B"; echo $C'
echo "A=$B$A$B B=$A$B$A"; echo "C=$B$C$B"; echo $C
```

selfcopy1.c

```
char *program[] = {
"char *program[] = {",
"0 };",
"#include    <stdio.h>",
"#define QUOTE 042",
"#define NL 012",
"main()",
"{",
"    char **line;",
"    puts(program[0]);",
"    for (line= &program; *line; line++)",
"    {",
"        putchar(QUOTE);",
"        fputs(*line, stdout);",
"        putchar(QUOTE);",
"        putchar(',');",
"        putchar(NL);",
"    }",
"    for (line= &program[1]; *line; line++)",
"        puts(*line);",
"}",
0 };
#include <stdio.h>
#define QUOTE 042
#define NL 012
main()
{
    char **line;
    puts(program[0]);
    for (line= &program; *line; line++)
    {
        putchar(QUOTE);
        fputs(*line, stdout);
        putchar(QUOTE);
        putchar(',');
        putchar(NL);
    }
    for (line= &program[1]; *line; line++)
        puts(*line);
}
```

Fascinated by the problem, I rediscovered the given analysis before finding reference [1]. But the challenge of a minimal solution took me further to a more functional approach. Then the first step (assignment) has to be replaced by an actual parameter of a call. The following LISP program illustrates this:

selfcopy.lisp

```
((lambda (x) (list x (list (quote quote) x)))
(quote (lambda (x) (list x (list (quote quote) x)))))
```

However this very simple (1 line) LISP1.5 solution is not working for systems with a baroque

quoting mechanism (try Fransz Lisp!).

Even in the sh(I) language it is possible to program in a more functional style by using the eval mechanism and positional parameter settings. Well, try to find a shorter solution within a (V7) UNIX environment then the following 48 byte shell program:

selfcopy2.sh

```
set \' 'echo set \\$1 $1$2$1\;eval \$2';eval $2
```

Finally the given C program can be reduced dramatically by using the format facility of printf in a functional way:

selfcopy2.c

```
char p[]="char p[]=%c%s%c;%cmain(){printf(p,042,p,042,012,012);}%c";
main(){printf(p,042,p,042,012,012);}
```

References

[1] Paul Bratley and Jean Millo, Computer Recreations, Software-Practice and Experience, Vol 2, (1972)397-400.

Description of the 'bed' Binary Editor

A. Adamson

LERS, Paris

1. Introduction

The binary editor **bed** was written in April and July 1983, based on the design discussed in *Design of a Binary Editor* by A. Adamson (February 1983 (unpublished)). The program was written with several goals in mind, viz.

- to re-implement the original version of **bed**, written in early 1981 to run under UNIX V7 on a PDP-11/70. This version had several bugs and was not very powerful.
- to be flexible and powerful in the description of the data format.
- to be an example of a clean and structured program.
- to be portable, with all machine dependencies left up to the compiler, instead of being in the program.

All of these goals have been achieved to a greater or lesser extent. This document will try to describe the program in terms of these goals.

The original version of **bed** was machine dependent, all structures had to be re-defined each time and could not handle *loops* (repetitions of sub-structures), and was well known to 'core dump'.

The port of programs and data from the LERS PDP-11/70 to a VAX-11/780 at the end of 1982 demanded a binary data conversion tool. To this end, the program **bconv(1L)**was The power of the format description language and in particular the possibility to define loops led to its adoption as a starting point for the data description language used by **bed**. Although the program was only used (to the best of my knowledge) three times during the port, the program enables the conversion of binary data in the two directions, and the basic principle was proved to be good.

This paper is a shortened version of the LERS internal report *The BED Binary Editor Version 2.1*. Section 6, entitled *Implementation*, has not been included in this report, although references to it have not been suppressed.

2. Language

The commands that manipulate the binary data need a description of the object(s) they are to use and where they are to do so. This section gives a formal definition and description of the language used by the binary editor to allow this to be done.

It should be noted that all commands must be given entirely on one line of input, and thus **newline** is never an acceptable character*. Further, spaces and tabs may be given freely at *any* point in the language between two literals and/or meta-variables, although their use between contiguous literals (apart from data types) is strongly discouraged.

2.1. Basic Data Types

The basic data types try to reflect the basic types used in programs. There are one, two, four and eight byte formats plus strings. The size of a given type is not placed in the program, but is left to that value assigned by the C compiler for the given machine.

---

* The newline character *is* acceptable within the definition of a function (see section 2.5). This is the only exception.

† The syntax throughout the document is given in BNF, where bold text represents *literals*, text in diamond brackets *describes* literals (without enumerating the possible values), and other text shows *meta-variables*.

The following syntax† (Fig. 1) is used for the basic types.

```
type      ::= inttype | longtype | realtype |chartype
inttype   ::= h | d | o | u | x
longtype  ::= l | D | O | U | x
realtype  ::= f | F
chartype  ::= c | b | s | S | s( posint ) | S( posint )
```

Figure 1

For each data type, except **short**, there are multiple output printing formats. The **h** is a short integer. The other **inttypes** fit into the local word size. The formats are for *d*ecimal, *o*ctal, *u*nsigned and hexadecimal. Similarly for the **longtypes**, except that **l** and **D** are synonyms. The real types are **f** for float and **F** for double-precision float.

The character types pose more of a problem. Bytes may be printed in octal with **b** or as ASCII characters with **c**. All eight bits of the byte are considered. With the **c** format, if the value would not give a printable character, the format defaults to octal. The **s** and **S** formats are for strings. The basic string, **s**, is defined as a sequence of characters terminated by a null byte. The **S** format is similar, except that the total length of the string (including the null byte(s)) is guaranteed to be even. Thus, the string "ab\0" has length 3 with the **s** format, but length 4 with the **S** format (the fourth byte is simply skipped without being looked at). Fixed length strings can be defined by putting the length within round brackets‡, immediately following the **s** or **S**. Exactly that number of bytes are printed. The even-byte quality takes effect even if a length is given, so that "S(13)" will print exactly thirteen characters, but the fourteenth will be skipped.

2.2. Structure Definition

Accessing the information in the binary file is done either through the use of simple types such as **integer** and **float**, or via **structures**. A structure is a named list of objects describing its format. Each of these objects may be a simple type or another structure. A structure may contain references to other structures and/or sub-patterns which define loops. A loop is a format list with an associated repetition count which may be constant, use the value computed by a function, or use the value stored from some previously read (integer, short or byte) datum value. There is thus the possibility to define, and execute functions and to store values for later use as counts.

The syntax for a structure is given below (Fig. 2) where the meta-variable **type** is as defined in the previous section. As can be seen, nested definitions of structures are allowed. However, this nesting is in fact removed during the parse so that nested definitions become structure definitions in their own right; the nested definition is replaced by an *instance* of that structure instead. (This is further explained in the section dealing with parsing and "code generation".) The semantics of structures disallow the possibility of recursive definitions, i.e. the structure *abc* may not be defined within the

‡ The BNF definition of **posint** is given in the next section. Suffice it here to say that it is a small non-zero positive integer.

definition of *abc*.

```
struct      ::= name { fmtlist }
fmtlist     ::= format | format fmtlist
format      ::= element | subpat | #struct | *sname
subpat      ::= [ fmtlist ] count
element     ::= type fmt count | inttype fmt store | fmtchar | posnchar | prtchar

fmt         ::= { conv } | <empty>
conv        ::= field | prec | field prec
field       ::= numstr | - numstr
prec        ::= . numstr

count       ::= < register | < func | repeat
store       ::= > register | > <empty>
repeat      ::= posint | <empty>

fmtchar     ::= n | r | t | "string"
posnchar    ::= p
prtchar     ::= P | N

sname       ::= ( name )
name        ::= register | register name

posint      ::= <non-zero positive integer (<2^16)>
register    ::= <lower-case letter>
func        ::= <upper-case letter>
string      ::= <any sequence of chars except RETURN (<100)>
numstr      ::= <string of digits>
```

Figure 2

As the # character is used to indicate a structure definition, the character * is used to show an instance (call) of the structure. For the user's convenience, an exception (the only one) has been made to the above grammar which allows the use of "*name" instead of "*(name)" as a legal **format**; it is, however, at the user's own responsibility to put a blank ot tab at the end of the name, if necessary, to avoid ambiguity with the succeeding structure definition.

The **store** met-variable enables the contents of the current "integer" type to be kept in the named register. This value can later be used (see **count**) as a repetition count. Note that an instance of a structure may *not* have a count associated with it; the effect can be achieved by placing the instance inside square brackets. The justification for this restriction is that the count factor for a format is kept with its (internal) description, and is thus operational for each "execution" of that format. Thus, a delete or append command would, by default, affect the structure of the given number of times, which could be zero or very many. The user of the editor would not intuitively know the number of times the structure was deleted or to be appended. By keeping the count to one (unless explicitly stated to the contrary), the user is avoided unexpected surprises.

The **fmtchar** and **poschar** meta-variables enable a certain freedom in the output formatting of a structure. Also, each simple type may have its own field and precision format (see **fmt**). The

ptrchar feature enables parts of records to be or not to be printed. This is described fully in the next section.

Only the first eight letters of a **name** are significant, although names may be as long as wanted. Strings must be less than 100 characters long, and cannot contain newlines. If the value of a count register or the return value of a function (at run time) is negative, this is treated as an error, and the command is aborted. A value of zero is tolerated as a repetition value, but cannot be given as a *constant* (i.e. **repeat**) value, since its effect would obviously be null. The value of a **posint** may be given in either decimal or octal. Octal numbers always start with a leading zero. Conversely, all numbers starting with a zero are treated as being in octal; the digits '8' and '9' are not allowed.

2.3. Output Format

By default, when no output formatting information is given, each item of a structure or repeated basic type is output one after the other, with two spaces between each item. These two spaces cannot be suppressed. Before printing the data, the current address is printed at the start of the line, followed by some space and a colon.

The **fmtchar** and **posnchar** are there to give some control on the overall formatting of the output. The **n** causes a newline to be output, plus sufficient white space to align the start of the data with that of the first line. The **r** character causes a space to be output, in excess to the two already there. A **t** causes a tab to be output. A string in double quotes is output as is, except that the quotes are stripped. Finally, the file position can be output with **p** which also lines up with the first line before continuing to output data.

The naming of these formatting characters comes directly from that of **adb**(1). The use of these characters can help greatly in the readability of the output of a binary file. It is unfortunate that the good use of such should be so little intuitive.

Data items may have individual formats associated with them (see **fmt**). Such formats may give a *field* size (with optional right/left adjustment indicator) and/or a *precision* indication. the meanings of these are as for **printf**(3).

If a record is large, it may be undesireable to print it out completely. The **prtchar** feature gives fine control over the printing of individual elements. The **N** turns off printing until a **P** is found, which turns it on again. Printing is always *on* at the start of a structure, though it may of course be turned off immediately.

2.4. Addresses

Certain commands allow for one or two addresses to be given. The current position is always used if no address is given, for such commands. If one address is given, the current position is changed to that address before the command is executed. If two addresses are given, the current position becomes that of the first address before command execution.

The syntax for addresses is given below (Fig. 3) where **sname** and **register** are as defined above. The **saddr**case and the **sname** possibility gives relative addressing, according to the (dynamic) size of the structure named.

```
maddr   ::= addr | addr , addr
addr    ::= saddr | saddr sname | $
saddr   ::= longint | 'register | <empty>

longint ::= <positive long integer>
```

Figure 3

The address *dollar* (**$**) gives the end-of-file (EOF) position. The semantics only allow the address *dollar* with certain commands, as indicated in section 3. When two addresses may be given, an empty **addr** before the comma defaults to address 0 and an empty **addr** after the comma defaults to address *dollar*. In all cases, when two addresses are given, e.g. *a,b*, the command is executed on the byte space between the addresses inclusive of the first and exclusive of the second, i.e. on the interval "[a,b)".

It is illegal to give an address which is negative, or that would cause the file pointer to exceed the end-of-file.

2.5. Expressions

When a command such as print or change is executed, the editor has to know on what the command is to be done. The *what* obviously involves a format (basic type or structure), but may also include a count, which is *not* part of the format itself. This combination of format and count is called an **expression**. When no expression is given explicitly, the previously given one is used. This expression is known as the **global** one. It is also useful for certain commands, to have an expression which is purely **local** to that command. For example, a structure is wanted to be printed but when the command is given, it is found that the start address was incorrect (e.g. one byte too few). The local expression can be used to increment the current position (by "+b"), and the print command can be regiven; the global expression was not affected by the "+" command.

Some commands set the definition of the global expression, others use this definition. Yet other commands expect a local expression, which defaults to the global one Finally, several commands have absolutely nothing to do with the expressions. The effect of expressions on commands and vice versa is fully described in the section on commands.

The syntax below (Fig. 4) shows how expressions may be constructed. This syntax applies only to local expressions for the same reason as explained in the section on structure definition as to why **sname** may not take a count factor. This apparent defect in the design should not cause alarm; it gives an intuitive consistency to the use and implementation of the binary editor.

```
expr    ::= *mname | mtype | repeat | <empty>
mname   ::= name | name repeat
mtype   ::= type | type repeat
```

Figure 4

Note that **mname** allows a count for a structure reference. The use of this feature is ill-advised under normal circumstances, since the output of the structures is contiguous; there is no way to include formatting in the definition of **mname** itself. The better way is to define another structure containing the multiple structures as a loop and the desired "punctuation".

2.6. Functions

As can be seen from the definition of **count** in Fig. 2 above, the value of a repetition count may be read from a **register** or evaluated by a **function**. The name of a function is a single upper-case letter. The following syntax (Fig. 5) details how a function amy be defined using the **&** command. **register** and **func** are as defined above. The value of the function (i.e. the value returned as the count) is

that of the last statement in the function. The value of the statement is that of the register, if assignment takes place, or that of the entire expression (after evaluation), otherwise.

```
function    ::= func { statelist }
statelist   ::= state | state ; statelist | state \n statelist
state       ::= register := fexpr | fexpr
fexpr       ::= sexpr | sexpr ? sexpr : sexpr
sexpr       ::= term | term binop term
term        ::= factor | uniop factor
factor      ::= register | func | integer | ( fexpr )

binop       ::= * | / | % | + | - | & | | | ^ | == | != | < | <= | > | >=
uniop       ::= _ | ~
integer     ::= <positive integer>
```

Figure 5

The operator '_' is for unary minus. The unary operators (**uniop**) have highest priority. the binary operators (**binop**) have priority decreasing as indicated below*, but have higher priority than the tertiary conditional operator (see **fexpr**).

```
* / %
+ -
== != < <= > >=
& | ^
```

There are possibilities of getting run-time errors when functions are used. These are caused by :-
attempted division by zero, a call to an undefined function,

- stack underflow (very unlikely), or stack overflow

The syntax allows for nested function calls, and thus also allows recursivity. However, it s the user's responsibility to ensure that recursive functions will terminate, otherwise stack overflow is guaranteed. As an example of recursion, the following function definitions will enable the sum of the integers from 1 to 9 to be calculated.

```
I { s := 0; i:= 9 }
C { s:= s + i; i := i - 1 }
T { C ? T : s }
```

The function **I**handles **s** to hold the sum, and **i** as for the loop control. The function **C** does the summing and gives loop termination, since the value of **i** is returned. Finally, the recursive function **T** loops while the condition **C** is true (with the summing as side-effect) and the sum is returned/printed when the loop ends.

* The operators on the same line have equal precedence and are evaluated left to right.

3. Commands

In the following descriptions of the available commands to **bed**, the column marked **A** shows whether ot not addresses are acceptable with the command. The character "a" means 0 or 1 address is allowed; "A" means 0, 1 or 2 addresses may be given; and "2" means 2 addresses *must* be given.

The column marked **P** indicates that the current data structure is printed, if it contains "P", or that the user has control over printing, if it contains "p". In this latter case, the command may be terminated with the letter "p" if printing is wanted.

The column marked with an **E** shows the interaction between the command and the data expression. A "G" means that the global expression is set by this command. A "g" means that the local expression is used. An "L" means that the expression is local to the command, the global one is being used, by default, if none is explicitly given.

The column marked with a ^ applies to the internal addressing. The current position in the data file is called dot (.). Commands that print the data expression set another internal address called hat () which is the position in the data file at the *end* of the print, i.e. hat = dot + expression. Note that dot, in this case, is not necessarily the position at the start of the command, but is the position before the start of the print.

COMMAND	A	P	E	^	DESCRIPTION
	a			^	An address on its own changes . and ^ to the addressed position in the data file.
# struct			G		Define a structure, using the definition given by *struct* (see section 2.2).
& function					Define a function, from the definition of such given above (section 2.6).
* mname	A	P	G	^	Print the structures(s) indicated by the expression *mname* (see 2.5 above).
/ mtype	A	P	G	^	Print the basic type(s) according to *mtype* (section 2.5). If no address is given, . is changed to ^ before the print takes place. Thus, the data file can be printed progressing continually through the file, and changing the format when necessary.
? mtype	a	P	G	^	This is similar to / except that . is not changed before the print. Thus, data from the current position may be re-viewed in a different format.
p	A	P	g	^	Print according to the global expression. . is not moved before the print. Useful to see the result of a change or append.
RETURN		P	g	^	The character RETURN without an address prints according to the global expression, but changes . to ^ before-hand. Used to print the file progressively without changing the format.

COMMAND	A	P	E	^	DESCRIPTION
a	a		L		Append according to *expr* (section 2.5). During the append, each item of the structure or each basic element is prompted with the code letter for the basic type and an arrow. The value should then be typed in. If no address is given, . is changed to ^ first, to simulate the append *after* the current object; otherwise the **a** and **i** commands are identical. The special address **$** for end-of-file may be given for this command.
i expr	a		L		Insert according to *expr*. This command is the same as **a** except when no address is given, in which case append moves to the end of the current object first, and insert adds at the current address. Note that to append/insert to an empty file, **$a** or **$i** must be used and not **0a** or **0i**. This is because 0 is an actual address and it is at the end-of-file for an empty file, and the EOF is never a legal address, as explained in section 2.4. The address **$** is special since it has no constant value, but rather represents a position.
d expr	a		L		OR
d	2				The first version deletes according to *expr*; the second deletes the byte space addressed (exclusive of the 2nd address). The data is deleted, but . is not changed. It will therefore point to the first datum after the portion deleted. In the second version, the second address may be **$**.
c expr	a		L		Change the data values according to *expr*. A change is in fact a delete or an append. . does not change.
m addr	2				Move the addressed byte space to the position indicated by (third) **addr** after the command character **m**. Either of the 2nd or 3rd addresses may be **$**. The third address may not be within the range of bytes to be moved. . takes the value of the third address when the command is completed. Note that there is a maximum of 32 blocks (i.e. 16384 bytes) which can be moved at a time.
t addr	2				Copy he addressed byte space to the position given by the third address. There is no restriction on the address except that the first address must be smaller than the second. . takes the value of the third address. The same size limit as for the **m** command applies here.
k register	a				Mark the current position. The value of . is stored in the named *register* and can be later accessed with the ' address syntax.
.					Print the value of ..
^		p		^	Move to the address ^. If a "p" (print) flag was appended to the command, printing will commence from this new position, and thus ^ will be reset to the end of the region printed. Only one object is printed.

COMMAND	A	P	E	^	DESCRIPTION
+ expr	a	p	L	^	Increment the file pointer according to *expr* (see section 2.5). If no "p" flag is given, the new position in the data file is printed and ^ is not changed. If the "p" lag is given, the increment takes place and then the print is done, changing ^ to the end of the zone printed. Note that the expression used to increment the file pointer is local and that only one object is ever printed.
- expr	a	p	L	^	Decrement the file pointer according to *expr*. If no "p" flag is given, the new position is printed and ^ is not altered. If the "p" flag is given, the change in address precedes the print. Only one object is ever printed.
r [file]					Redirect the program's input from the named file. At the end of the file, the input resorts to the previous input stream. Up to five input streams may be nested at any one time. If no file name is given the standard input is accepted.
w [file]					Write the edit file to the named file. If no name is given, rewrite the new contents onto itself. It is an error to try to rewrite onto the edit file if the contents have not been changed since the edit started or the last write.
W [file]					Force a write to the named file, or the edit file. This really only makes sense to rewrite the edit file. The advantage is that the *cuts* are re-initialised (see below).
e [file]					Change the edit file to the named one, else itself (i.e. re-initialise). It is not allowed to re-edit a file if the current edit file has been changed since the start or last write.
E [file]					Force a change of edit file to the named file, or itself.
q					Quit the editor. This is only possible if the edit file has not been modified since the start of edit or last write.
Q					Force a quite of the editor.
" state					Execute the given statement from the syntax given above (section 2.6). The value of the executed statement is printed on the following line, indented and after an equals (=) sign.
% c					Give information on some part of the existing internal values according to the control character *c* which is one of:-
					s [name] Print out the structure named, or if no name is given, print out all structure definitions. The print-out is in the same format as the definition was originally given, except that field/precision formats are not shown.
					S Print out all structure definitions in detail. Useful for debugging.

COMMAND	A P E ^	DESCRIPTION
	i	The name of the edit file, its current size, the current values of . and ^, and the current global expression (if defined) are printed.
	c	The same information as with **i** (except for the global expression) is printed along with th details of the internal situation as regards the *cuts* of the file (see below under "Implementation" for details). The current cut is marked with an asterisk. Useful only for debugging.
	r	The values of all non-zero structure registers are printed, one per line in the format 'register : value'.
	a	The values of all kept addresses (see **k** command) are printed in the same format as for **r**.
	f	The definitions of all functions are printed, one per line. Note that the function body is given in a postfix (Reverse Polish) notation.
! command		The *command* is passed to the *shell* (sh(1)) to be executed. All characters "!" within the shell command are substituted for with the previous command (if any) before being passed to the shell.

If an error is found in a command or an illegal address is given, the error is reported and the command aborted. Syntax/semantics errors are noted by an up-arrow under the character where the fault was found, if the input is the standard input. If the input is from a file, the file name, line and character positions are given.

4. INVOCATION

The call to the binary editor has the following syntax.

bed [-] [-i inputfile] [editfile]

The - option is for silent mode. In silent mode, the message giving filename and size (on startup and with the "e" command), the message about creation of the edit file, and messages about structure redefinition are suppressed.

The **-i** option can be used to tell the editor to read its input from the named **inputfile** instead of the (usual) standard input. Note that this is *not* the same as redirecting the standard input, since the editor distinguishes between the two when indicating syntax errors.

Normally an **editfile** will be given, and this file is opened (created if necessary) at the start. If no file is stated, the file "bed.out" is created. If the file "bed.out" is used and nothing is put into it, it is removed when the "q" (quit) command is given.

All prompts and error messages are sent to the standard error stream *stderr,* so that such can easily be caught and sent elsewhere if desired. On the other hand, output caused by user-given commands is sent to the standard output stream *stdout.*

Normal exit status of the program is zero. Normal exit is via the q and Q commands. If the end-of-input is found (see sub-section 6.4.1), the program terminates with exit status one. A fatal error (see section 6.5) causes termination with exit status two.

5. PROGRAMMING

The binary editor **bed** is entirely written in the C language and is intended to be portable, and thus doesn't use "additional features" such as structure assignment. However, the program does make use of the *enumeration* type. This should not prove much of a problem, if a given C compiler does not accept them, since all such code is well distinguishable, and such code could easily be changed to appropriate #*define* expressions. The files affected are "type.i", "cut.i", "elem.i" and "kstack.i".

5.1. Programming Philosophy

The program was written in a novel style which tries to make clear the dependencies and interactions of global variables by creating distinct and "visible" interfaces between modules.

The basic assumptions are simple.

1. A module is self-contained.
2. The outside world can "use" the module via, and only via, that module's interface. The "use" can involve actions (function calls) and/or values (global variables). The outside world cannot change the value of a module's variables.
3. The implementation of a given module is totally transparent to the outside world.

The scheme is implemented by using ".h" files* to describe the interfaces. To keep in line with the third assumption, implementation details which are not directly part of the interface, but which are necessary for the program to compile, are hidden at another level in ".i" files.

The modules are in ".c" files and generally include quite a lot of ".h" files. When implementation details which do not form part of an interface are needed by a module, the ".i" file is directly included.

Looking at it from another angle, ".c" files contain functions. Global variables and functions not forming part of the interface in these files are marked *static* to show they are private. Access to other modules is via "h" files. Access to implementation details of a general nature is through ".i" files. Thus, a ".c" file can be read, *independently* from all other ".c" files; only the ".h" and ".i" files that are included in that ".c" file need to be read to have a complete picture.

Interfaces are in ".h" files. These contain declarations of functions and/or variables constituting the interface. If the variables' implementation is irrelevant to the interface, these will be in a ".i" file which is included in the ".h" file.

The ".i" files hide all implementation details for interfaces. Further, general implementation details, such as structure definitions, which need to be available in several places, are in ".i" files which are included directly in the ".c" files concerned.

5.2. Programming Practice

Theory is all very well but putting it into practice usually causes its own problems. The binary editor was no exception.

A module often has to split into sub-modules. Thus the interface scheme has also a (theoretical) *levelling* associated. However, every now and then, some detail from a lower level is needed in a higher level. This causes some interface to be included where it shouldn't really be.

The same applies to individual implementation details which really private to one (sub-)module, are found found to be necessary elsewhere. In such cases, the global (private) variable is not marked *static* and its use elsewhere is clearly marked with *extern* and a comment stating where it really is. This is kept to a strict minimum.

A module may provide several non-overlapping interfaces. This is the case when initialisation and use are to be dealt with in separate parts of the program. The problem is really only one of naming (since a module *never* includes its own interface description).

---

* ".h" files are files whose names end in ".h".

To conclude, the theory has been adhered to as far as possible and has proved to be beneficial to the modification, comprehension and reading of the program which is fairly large (just short of 5000 lines of code).

5.3. Programming in C

The major drawbacks to programming in C are that of **typing** objects and **declaring** them. For variables, *typedefs* are used a lot to try and clarify the underlying intention and use. When *int* is used, it is implicit that the object has no intrinsic size in bytes apart from that of the machine on which it is running. For functions, the distinction between **functions** and **procedures** is made using the *void* feature. Since the *void* type does not exist under V7 UNIX, this is defined as *int* for that system.

There is a problem with functions which is evident in the parsing routines. Only a few of the parsing routines are called explicitly from elsewhere, and these should therefore be in an interface. However, all the parse routines are homogeneous in their task, and it seems ridiculous to mark some as being part of an interface and others as being private to the parser concerned. In this case, none of the functions are marked and no interface file is used.

5.4. Compilation

The **Makefile** not only tells **make** how to do its job but also gives the user information as to the internal set up and use of the files. The rule **.i.h** is used to show the dependencies between ".h" and ".i" files.

The standard C library causes problems for string handling, hence the files **string.c** and **string.h**. Routines from my local library are also used, but this has the undesirable side-effect of making the program less portable to other sites, since the library has to be moved as well.

Two programs, many UNIX systems (reprint)

Andrew S. Tanenbaum
Vrije Universiteit, Amsterdam

Teus Hagen
Mathematical Centre, Amsterdam

UNIX meetings gives a splendid opportunity to run test programs on the machines present at the exhibition. At the recent meetings and exhibitions in Europe and the US, we have run two test programs on a wide variety of machines. Test program #1 measures CPU/memory speed; test program #2 measures I/O speed. Program #1 was tested six times, with the 'TYPE' declared in six different ways: short, register short, int, register int, long, register long. On the small machines, the test were generally made in single user mode; on the large mainframes we had the share the machine with other users.

The programs:

```
/* Test 1 - CPU/memory */
main()
{ TYPE i, j, k;
  for (i = 0; i < 1000; i++)
        for (j = 0; j < 1000; j++)
              k = i + j + 1983;
}
```

```
/* Test 2 - I/O */
main()
{ int i, j, n;
        char a[512];
        if ( (n=creat("foo",0755)) < 0)
              perror("error: foo");
  for (i = 0; i < 500; i++)
        write(n, a, 512);
}
```

Notes:

The times reflect a combination of several factors, among them, the CPU type, the clock rate, the speed of the memory management unit, the speed of the memory itself, the width and speed of the bus, and last, but certainly not least, the quality of the C compiler used on the machine. Also, the times were obtained using the time(I) command. There is reason to believe that not all vendors understand that 50 Hz != 60 Hz, which makes some of the times slightly suspect.

Conclusions:

None. You should take these measurements with a grain, or better yet, an imperial gallon, of salt. For example, comparing the PDP-11/70 with the SUN, we see that for test #1 and register short, the PDP-11/70 is nearly three times faster, but comparing register long for the same two machines, the SUN is twice as fast. The difference can be explained by the fact that the PDP-11/70 really is faster, but uses memory instead of registers for register longs, whereas the the SUN uses the 68000's hardware registers.

Goal:

Our goal in making these measurements is to stimulate you into making your own measurements and to make you cautious when looking at (carefully selected) comparisons thoughtfully supplied by vendors. Remember: Figures don't lie, but liars figure. If anyone wants to run the tests on other machines, we would appreciate hearing from you.

times in seconds: machine	MHz	usr short	usr short reg	usr int	usr int reg	usr long	usr long reg	real cc#1	real cc#2	#2	year	rem
DEC:												
VAX 780		8.8	8.6	6.7	4.7	6.7	4.7	3	3	2	83	
VAX 750		16.2	16.5	10.9	7.3	10.8	7.3	11			84	
VAX 750		18.9	19.3	13.0	8.5	11.6	8.6	4	4	2	83	
MicroVAX		28.2	28.3	22.2	12.4	22.2	12.4	12			84	
VAX 730		37.4	37.5	23.9	14.4	24.1	14.4	11	12	6	83	
11/70		7.4	2.8	7.4	2.8	14.3	14.3	11	12	14	83	
11/60		10.4	4.2	10.4	4.2	20.2	20.2	17	19	16	83	
11/44		11.2	5.7	11.2	5.7	21.8	21.8	16	8	21	83	
11/45		19.1	7.9	19.1	7.8	38.5	38.5	12	20	12	83	
11/34		22.0	10.9	22.0	10.8	44.8	44.8	18	19	14	83	
11/24		27.7	12.2	27.4	12.4	57.0	57.5	21	20	19	83	
AEDS11(/23)		34.5	14.9	34.6	14.9	72.2	72.2	28	30	8	83	
micro 11(/23)		36.8	16.2	36.9	16.2	76.8	76.8	29	31	24	83	
68000's:												
Plexus P/35	12.5	8.9	5.3	10.0	5.1	10.1	5.2	12			84	
Ch Rivers	12.5	10.1	6.0	11.8	5.9	11.8	5.9	12			83	
QU68030	10	12.0	6.0	13.0	6.0	17.0	8.0	8	11	4	83	
Parallel	12.5	11.6	7.0	13.5	6.7	13.5	6.7	17	18	8	83	
Altos	8	13.9	13.9	13.9	13.9	17.8	17.8	18	25	5	83	
SUN 1	10	13.0	7.8	14.6	7.6	14.6	7.5	9	10	5	83	
Cyb	8	15.3	9.0	15.3	8.2	17.5	8.2	27	28	12	83	
Momentum 32E	8	14.5	8.6	16.7	7.6	16.7	7.6	22			83	
TI NU	10	13.8	6.7	18.2	6.5	17.7	6.6	16			84	3 usrs
Codata	8	17.1	10.5	19.3	10.5	19.3	10.5	28	30	28	83	
CCI	8	20.5	9.3	20.5	9.3	29.9	13.2	13			84	
Pacific	10	18.1	10.7	20.7	9.7	20.8	9.7	24	23	13	83	
Power 520	8	21.1	9.2	20.7	9.2	31.4	12.5	16	16	19	83	
Hawk 32/E	8	19.0	11.3	21.5	10.3	21.5	10.3	29	19	27	83	
Pixel 100/AP	8	18.6	11.0	21.5	9.6	22.9	9.6	44	47	10	83	
Corvus	8	19.8	11.6	22.5	10.1	22.6	10.1	40	42	no sp	83	
QU68000	10	24.5	11.9	24.5		33.0	15.1	10	10	7	83	
Fortune	8	21.7	12.8	25.0	12.4	25.0	12.3	18	20	6	83	
Apple Unisoft	5	22.6	13.7	25.9	12.1	25.9	12.1	41	43	28	83	
Apple Xenix	5	22.7	13.9	26.0	13.0	25.9	13.0	58	65	no sp	83	
Altos-12	5	26.7	14.1	26.7	14.1	53.7	53.9	31	33	13	83	
Plessey S68	8	23.9	13.9	27.5	12.8	27.5	12.6	30			83	
Wicat WS150	8	24.8	14.4	28.1	13.1	28.1	13.1	19	21	12	83	
Victory 68K	10	23.7	15.0	28.3	12.6	29.1	12.5	19			83	
TRS80	8	25.2	14.9	28.3	14.0	28.4	14.2	23	28	16	83	
Codata	8	25.5	14.9	29.2	12.8	29.2	12.8	16			84	
Dual	8	26.9	15.6	30.7	13.3	30.7	13.3	21	20	27	83	
Four Phase	8	27.4	16.2	31.1	14.9	31.1	14.9	32			83	
Ch Rivers	8	28.2	15.8	31.7	15.8	31.7	15.8	28	42	14	83	
Unistar 200	8	28.7	16.5	32.8	14.3	32.8	14.3	36	40	28	83	
IBM Sritek	8	30.3	17.6	34.2	17.3	35.1	16.9	31	35	18	83	ut>rt?
IBM PC Idris	8	37.9	22.7	37.9	22.7	73.3	73.3	18	26	39	83	
Cosmos Antaris	8	34.1	19.5	38.7	16.4	38.7	16.4	26	27	9	83	
ULAB	8	37.2	21.0	44.1	18.9	44.1	19.0	38	44	15	83	

times in seconds: machine	MHz	usr short	usr short reg	usr int	usr int reg	usr long	usr long reg	real cc#1	real cc#2	#2	year	rem
Z8000's:												
Z6000	5.5	13.6	6.3	13.6	6.2	23.2	12.6	20			83	
Zilog	6	14.7	7.3	14.7		25.7	13.3	20	20	8	83	
Plexus	5	15.2	7.0	15.4	7.0	27.5	27.6	24	26	13	83	
ONYX	4	15.9	7.2	15.9	7.3	23.8	14.1	14	14	8	83	
Bleasdale	4	33.3	15.6	33.3		56.2	56.2	20	21	16	83	
8086's:												
Altos	10	13.7	7.2	13.7	7.2	27.8	27.7	18	20	7	83	
Intel	8	29.2	17.6	29.0	17.5	59.1	59.0	17			84	
SBC 86/12A			68.1	83								
8088's:												
IBM XT	4	53.4	30.0	53.4	30.0	108.8	108.8	25			84	
286's:												
Intel	5.5	17.3	10.9	17.3	10.7	34.6	34.6	27			84	
16032's:												
National	6	27.4	25.6	29.3	14.0	32.4	11.7	20			83	
IBM Sritek	6	27.4	27.5	32.1	13.4	30.7	13.6	103			84	
National	4	49.9	45.0	56.7	23.3	57.4	27.2	47	49		83	mem flt
others:												
Amdahl		0.5	0.5	0.3	0.3	0.3	0.3	10	5	1	84	1 usr
Concept32/87		1.5	1.5	0.9	1.4	1.0	1.4	10	16	2	83	1 usr
Pyramid		3.3	3.3	2.0	1.9	2.0	2.0	9			84	
Ridge		7.1	2.9	4.0	1.6	4.0	1.6	19			84	
Eclipse		4.6	4.6	4.4	4.4	4.4	4.4	28			84	
Arete		5.7	5.7	5.7	5.7	5.6	5.8	15			84	
HP 9000		9.4	9.4	7.4	7.4	7.4	7.4	28	26	3	83	
Concept32/27		12.0	11.0	10.0	10.0	10.0	10.0	18	20	5	83	
BBN C/60		14.5	8.2	14.6	8.2	47.2	30.3	7	9	3	83	
PE3210		16.7	6.7	15.9		15.9	6.7	4	4	3	83	
Perq 1		44.5	15.6	22.2			15.0	25	25	7	83	
Perq 2		46.7	15.1	23.0	15.0	22.9	15.1	20			83	
IBM S/1 4954		37.2	37.1	37.1	37.1	62.3	62.3	28	30	32	83	

Circular UNIX made trivial

Jaap Akkerhuis

Centrum voor Wiskunde en Informatica
Amsterdam

"Yes", is the answer to Timothy Murphy's question or there is a way of making "personalised" circulars or form letters under UNIX.† Although it is fun to (ab-)use the C-preprocessor for this problem, the obvious choice is of course N/Troff. These nice programs have the `.rd` request, which will perform the function you want, in combination with the `.nx` request. Suppose you have some small macros defined, to have a standard layout. Let's take the next one's

```
.OD         - print the current date
.AD         - start of the adress
.AE         - end of the adress, the adress will be print, so a
              window envelope will show it
.A "hello"      - print out the string hello as the greeting message
.B          - start of the body of the text
.E          - end of the text, do some indentation
.RT         - reset indentation etc, to the default state
```

Now we construct one file with the letter called "letter", what will look like:

```
.OD
.AD
.rd             \" Read the data for the person
.AE
.A "Dear \\*(gr,"
.B
I don't want to reinvent the circle.
.E
Sincerly yours,
.sp 2
Jaap Akkerhuis
.RT
.bp 1               \" generate next page with pagenumber 1
.nx letter          \" process the file again.
```

The "personal" data is in the file "persons" and will look like:

```
.ds gr Timothy
Timothy Murphy
Trinity College
Dublin

.ds gr Jim
Editor of the EUUG Newsletter
Jim Mckie
Centrum voor Wiskune en Informatica
Kruislaan 413
1098 SJ Amsterdam

.ex
```

Now we call

```
[nt]roff marcos letter < persons
```

and the **.rd** request in the file "letter" will make [nt]roff read from standard input, which happens to be the file "persons". It will read the file up to the empty line (or more precisely, until two newlines in a row are found). Then the rest of the file letter is processed and the **.nx** request will cause it start all over again. The **.ex** (exit) request will finally stop all the processing.

As you see, it is all there with the standard UNIX tools.

Stichting Mathematisch Centrum
Kruislaan 413 1098 SJ Amsterdam

MC

Uw referentie
Datum
Onze referentie
Datum 6 januari 1984

Editor of the EUUG Newsletter
Jim Mckie
Centrum voor Wiskune en Informatica
Kruislaan 413
1098 SJ Amsterdam

Dear Jim

I don't want to reinvent the circle.

Sincerly yours,

Jaap Akkerhuis

Stichting Mathematisch Centrum
Kruislaan 413 1098 SJ Amsterdam

MC

Uw referentie
Datum
Onze referentie
Datum 6 januari 1984

Timothy Murphy
Trinity College
Dublin

Dear Timothy

I don't want to reinvent the circle.

Sincerly yours,

Jaap Akkerhuis

THE AMSTERDAM COMPILER KIT

Ed Keizer
Andrew S. Tanenbaum
Hans van Staveren

Dept. of Mathematics and Computer Science
Vrije Universiteit
Postbus 7161
1007 MC Amsterdam, The Netherlands

Telephone: 31 (20) 548 5392
UUCP: ..!mcvax!vu44!keie

1. INTRODUCTION

The cost of producing compilers for high-level languages is becoming one of the major cost factors in producing new systems. Traditionally, a separate compiler was produced for each combination of language and machine. The Amsterdam Compiler Kit is designed to ease the production of new compilers. This paper describes the components of the kit and some experiences with a few of the compilers we produced.

For each high-level language in our kit we have a program, called front end, that translates the source into an intermediate code, called EM. For each machine language in our kit whe have a program, called backend, that translates EM code into assembly language. The use of an intermediate language has several advantages:

- adding one front end for a new language allows use of that language on several machines.
- adding a back end for a new machine allows the use of several high-level languages.
- it is possible to use optimizers on the intermediate code, thereby lifting a burden from the shoulders of the front end writer.

One of the design goals of the architecture of our intermediate language was to ease the task of the front end writer.

The idea of producing compilers using a common intermediate code (often called an UNCOL) is hardly new. What we have done is work out the details and actually make a practical implementation that runs on UNIX and produces high-quality compilers for a variety of languages and machines.

In the following sections we will describe the various components of the tool kit in some detail. These are shown in Figure 1.

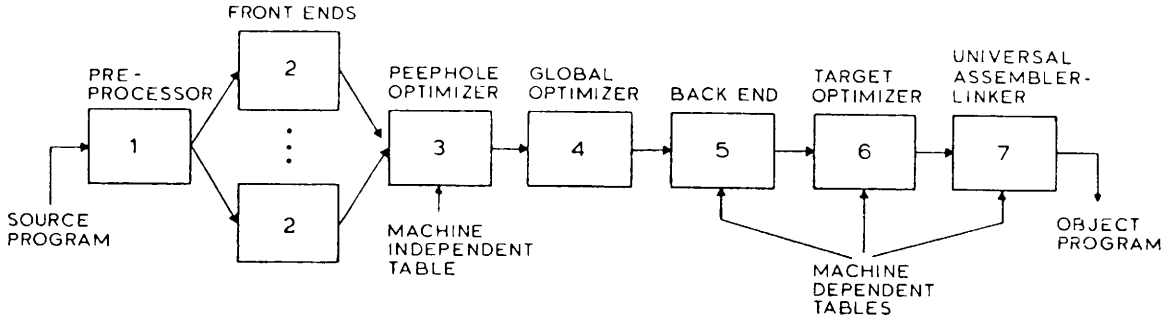


Fig. 1. The Amsterdam Compiler Kit (ACK).

2. THE PREPROCESSOR

The preprocessor is the standard C preprocessor used by the UNIX compilers. It handles textual inclusion of header files, macro expansion, and conditional compilation, making these facilities available in all the languages in the tool kit.

3. THE FRONT ENDS

The front end converts the preprocessed source program into EM, which is a simple stack machine with few of the idiosyncracies of conventional machines. Producing EM code for expressions, for example, is not much harder than producing reverse Polish. As an example of EM code, consider the following C statements, in which the letters I, J, and K all designate local integer variables:

```
I =  5*J + K;
F(I, J-3);
```

The corresponding EM code for an implementation with 2-byte integers is as follows, where semicolon is used to introduce comments:

```
LOC 5        ; PUSH CONSTANT 5 ONTO THE STACK
LOL J        ; PUSH J ONTO THE STACK
MLI 2        ; MULTIPLY INTEGERS OF LENGTH 2 BYTES
LOL K        ; PUSH K ONTO THE STACK
ADI 2        ; ADD INTEGERS OF LENGTH 2 BYTES
STL I        ; STORE THE TOP OF THE STACK IN I
LOL J        ; PUSH J ONTO THE STACK
LOC 3        ; PUSH CONSTANT 3 ONTO THE STACK
SBI 2        ; COMPUTE J-3
LOL I        ; PUSH I -- NOTE: C CALLING SEQUENCE PUSHES ARGS BACKWARDS
CAL F        ; CALL THE FUNCTION
ASP 4        ; ADJUST STACK POINTER BY 4 BYTES TO REMOVE THE ARGUMENTS
```

Each front end is written as a separate program. The Pascal front end, for example, is written in Pascal itself and uses a recursive descent parser. The C, front end, in contrast, uses a bottom up technique. The calling sequences for procedures in both languages are so similar that programs can be written in a mixture of both languages.

The EM code can either be further processed for compilation to target machine language or assembled to a binary form for interpretation. The latter route speeds up compilation and and is mostly used during program development since the interpreters can check for run-time errors and gather detailed execution statistics.

4. PEEPHOLE OPTIMIZATION

The local, or peephole, optimizer reads the EM file and produces a new, improved EM file. It maintains a small window on the file in which it looks for the patterns present in its driving table. When a pattern is found, the replacement text is also taken from the driving table.

The approximately 400 optimizations present in the driving table can be roughly categorized into the following major groups:

1. Constant folding
2. Strength reduction
3. Reordering of expressions
4. Removal of unreachable code
5. Removal of null instructions
6. Group moves
7. Use of special instructions

In addition, the peephole optimizer also does branch chain compression and a few other optimizations that are not strictly table driven.

By putting all the peephole optimization in one place, we have an optimizer that can be used with all languages and all machines in the tool kit, eliminating the need for each compiler writer to do his own optimization.

5. THE BACK END

In contrast to the front ends, which are a collection of distinct programs, the back end is a single table-driven program. It reads the EM file and generates target assembly code by simulating at compile time the behavior of the run-time stack. The driving table for a given machine has a series of lines, each one having five parts:

The EM instruction to be translated
The stack configuration when the line can be applied
The code to be generated
The new value of the stack after the line has been applied
The time and space costs of applying the line and generating its code

As each EM instruction is read, the table is searched for lines that apply to it. For example, the EM code for I = J + 3 , consists of LOL J followed by LOC 3 and then an addition. At the time of the addition instruction the simulated stack has a constant on top and a (local) variable just below. Thus the back end must find a line in its table telling what to do with an ADI 2 instruction in the context of a constant and a local variable. If such a line exists, it can carry it out.

On the other hand, if no such line exists, then it must find some other way to generate the code. For example, if there is a line telling how to do an addition with a register variable and a constant on the stack, it could then try to find a line telling how to get a variable into a register, and then apply the former line. In general, there will be several legal code sequences. In this example one could first load the constant into a register and then add the variable to it; alternatively, one could first load the variable and then add the constant to it. On some machines it may not matter, but on, for example, the TI 9900, a cheap instruction to move a small constant into a register exists, but no corresponding instruction exists to add a small constant to a register, so the first code sequence is better than the second one.

The code generation algorithm is now simple to describe. Try each of the legal code sequences. Each one has a cost and leads to a new stack configuration. From each of these configurations, a new EM instruction is read in and all the legal code sequences generated, and so on. In short, a weighted tree is built, not unlike a chess-playing program does. Using this tree, the first step along the most "efficient" path is taken, and the whole process repeated. The user may specify "efficient" to mean time or space or some linear combination thereof.

6. THE UNIVERSAL ASSEMBLER

The final component in the main path is the universal assembler. It also serves as the linker, reading in libraries and extracting routines where needed. Like the back end, it is parameterized by a machine-dependent driving table. It is surprising how machine-independent an assembler can be. Our experience is that a student who knows nothing about the tool kit or assemblers can produce an assembler for a machine such as the Z8000 in three weeks, and that an experienced programmer can write one in about one week.

7. INTERPRETERS

An alternative way of executing programs is to take the EM code from the front end and assemble it to a special, highly-compact binary representation and then interpret it. This approach yields faster compilation at the price of slower execution. Interpreters can also provide substantial run-time checking, debugging, and profiling information and are especially useful during the development phase. Currently, all our interpreters are written in the assembly language of the machine on which interpretation is to take place.

8. GLOBAL OPTIMIZER

The global optimizer is currently being developed, it will look at each program as a whole and try to find ways to speed up execution. Three distinct types of optimizations can be found here:

1. Interprocedural
2. Intraprocedural
3. Basic block

Interprocedural optimizations are those spanning procedure boundaries. The most important one is expanding procedures in line, especially short procedures that occur in loops and pass several parameters or procedures calls with constant actual parameters.

Intraprocedural optimizations take place after analysis of individual procedures. Much attention is given to loops inside a procedure. The global optimizer tries to move code outside the loop and attempts to allocate registers to variables used inside loops.

A basic block is a piece of code that can only be entered at the top and left at the bottom. The global optimizer tries to transform arithmetic or boolean expressions into forms that are likely to result in better target code. A simple example of this is transforming $-B + A < 0$ into the simpler $A < B$.

9. DISCUSSION

To get an idea of how well the compilers produced by the tool kit compare with other portable compilers, we compiled 73 programs from /usr/src/cmd using both our compiler and pcc on the PDP-11 and the VAX. This test did not use the -O flag of pcc, nor the analogous program in our system, the target optimizer. Furthermore, all the register declarations were replaced by auto declarations for both compilers to to compensate for the fact that our system currently stores all variables in memory.

The ratio of our code to pcc code on the PDP-11 varied from 0.87 (our code was smaller) to 1.28 (pcc code was smaller). The distribution was as follows:

ack size/pcc size	occurrence frequency
0.00-1.00	8%
1.01-1.05	20%
1.06-1.10	35%
1.11-1.15	24%
1.16-1.20	5%
1.21-1.25	4%
1.26-1.30	4%

On the average, our code was 9% larger than pcc code.

The ratio of our code to C-compiler code on the VAX varied from 0.91 (our code was smaller) to 1.21 (cc code was smaller). The distribution was as follows:

ack size/cc size	occurrence frequency
0.90-0.95	1.4%
0.95-1.00	2.7%
1.00-1.05	16.4%
1.05-1.10	35.6%
1.10-1.15	28.8%
1.15-1.20	12.3%
1.20-1.25	2.7%

On the average, our code was 9% larger than cc code.

Our system is still being tuned, and improvements will no doubt be generated in the future, especially when the global optimizer becomes available.

We also translated these 73 programs to Intel 8086 code. We could not compare the quality of that code with code produced by other 8086 compilers. We did compare the code size for the same programs on Intel 8086, VAX to the code produced by our compiler kit for the PDP11. The distributions were:

Intel 8086/PDP 11	occurrence frequency
0.70-0.75	1.7%
0.75-0.80	10.0%
0.80-0.85	41.7%
0.85-0.90	43.3%
0.90-0.95	3.3%

Intel 8086 code is, on the average, 17% smaller than PDP11 code.

Vax/PDP 11	occurrence frequency
0.70-0.85	4.2%
0.85-0.90	7.0%
0.90-0.95	8.5%
0.95-1.00	11.3%
1.00-1.05	14.1%
1.05-1.10	16.9%
1.10-1.15	14.1%
1.15-1.20	14.1%
1.20-1.25	5.6%
1.25-1.35	4.2%

The code for the VAX is, on the average, 5% larger than PDP11 code.

10. CURRENT STATUS

The current status of the project as of June 1983 is as follows:

Front ends: Pascal and C done; Plain and Algol 68 in progress
Peephole optimizer: done
Back ends: PDP-11, VAX, 68000, 8086 done; Z80, 8080 Z8000, 6809 in progress
Assemblers: 6800, 6809, 68000, 8080, Z80, Z8000, 6502, PDP-11 done
EM interpreters: Z80, PDP-11 done; 68000 in progress

The tool kit itself has been tested on the PDP-11 under UNIX V7, on a VAX 11/780 under Berkeley UNIX, and on a 68000 under System III. In principle, moving to other UNIX systems should not cost more than a week or two.

The tool kit will be licensed to universities holding an academic UNIX source license from Western Electric for a moderate charge (probably $500) starting in September 1983. This distribution will contain the complete sources of all the programs, and a large amount of additional information, including libraries, programs to test new back end tables, documentation etc. It will also be made available to corporations holding a UNIX source license for a licensing fee to be negotiated, depending on which rights the corporation desires (e.g., in-house use vs. sale to customers; national rights vs. world-wide rights, etc.). Interested parties should contact the authors.

MAPPING THE UUCP NETWORK

Rob Kolstad

CONVEX Computer Corporation

Karen Summers-Horton

The UUCP network encompasses those machines on the UNIX News Network (Usenet) and more: over 2100 sites as of January 11, 1984. This talk details the difference between the UUCP network and Usenet in addition to outlining the current problems in using the networks (notably those concerning reliable routing among sites: "how to get there from here"). If the UUCP network is to become an ARPA domain, reliable maps and site descriptions must be available to a "name server". This talk will also explain our current plans for mapping not only the connectivity of the network but also the "quality" of the connections for use with routing programs such as Steve Bellovin's optimal path finder. The talk will include current schema for collecting, disseminating, and using the information.

1. UUCP and Usenet

UUCP (Unix to Unix CoPy) is a file transport mechanism which also features the ability to request execution of a few commands at remote sites (e.g., rmail and rnews). It is used primarily to send electronic mail and news. The network is not always "connected": a host often calls another site only on a schedule or when traffic is waiting for the remote site. AT&T Bell Laboratories designed UUCP in 1976 with the assumption that all machines had an auto-dialer and could call all other machines directly (a valid assumption when there were only a few hosts running UUCP). Nowadays, each host connects to one or several remote machines. These hosts range in size from small microcomputers (e.g., Radio Shack TRS-80 model 16's) to giant multi-user systems.

UUCP and rmail use a clever idea to move mail to sites not directly connected to a host. The program "rmail" examines a mail header (which specifies sites connecting the original host and the mail's destination) and sends the mail along the first site listed on the list after discarding it. To send mail from site 'parsec' to user 'joe' at site 'adec23', a path like this emerges:

allegra!alberta!sask!hssg40!adec23!joe

Clearly, it is of utmost importance to know who talks to whom! These paths become more complicated when internetwork routes enter the picture. It is possible to send mail from the UUCP network to the ARPANET, DEC E-NET, BITNET, and many local networks. Today, over 2100 hosts run UUCP; this is where our problems begin.

A fraction of these machines (approximately one-third) also belong to a logical subnetwork called "Usenet". Usenet is an electronic bulletin board connecting about 700 sites in the United States, Canada, Europe, and Australia. A person at any Usenet site may post articles to any specific bulletin board (called a "newsgroup") and (eventually) reach all people who subscribe to that newsgroup.

News is transferred by various means: UUCP, local area networks, and long haul networks to name a few. Maintenance and development of Usenet software is done on a volunteer basis; the cost for transferring information is borne by each individual site. A Texas site pay $2,000 to $4,000/year to the phone company for long distance charges for news and mail. Some sites pay nothing: Southern Methodist University (with no budget) is polled by local industries. Digital Equipment Corporation maintains some overseas links; their phone bill is around $19,000 per month.

2. Problems with UUCP

Along with the advantages of low cost and ease of use, the UUCP network has problems. In summary, these problems are:

a. Explosive growth
b. Addressing reliability
c. Lack of backward error recovery
d. Unknown and unpredictable propagation delays

We will address the problems of growth and addressing here.

UUCP is growing by leaps and bounds. Our current guess of over 2100 hosts is just that: a guess. There is no official central registry of UUCP hosts. When someone joins UUCP, there is no official procedure. Once they find someone who will let them hook up to their site, they then can mail to the whole net without anyone except their immediate neighbor really knowing who they are. As a result, trying to get mail to John Smith at Framus, Inc. is difficult unless you happen to know the path through the network to the Framus computer. With over 2100 sites, (most of which are not directly connected to each other), this becomes nearly impossible. To find your way through the net, you need a road map. Our project is to provide that map.

Since the net is growing so quickly and since it is so easy to set up or tear down a link, the net's structure is never fixed. Not only must the initial map be collected, but it must be constantly maintained or it will be quickly out of date. This maintenance will entail an enormous effort.

Our current guess based on current growth patterns and computer sales is is that the net will increase to between 15,000 and 50,000 sites over the next five years. One reason is the increasing popularity of personal computers, whose owners will want the convenience and relative low cost of UUCP as an option. We must plan now for this explosion.

Once a map exists in machine-readable form, it is a simple matter to use that map to route mail. This means that instead of typing in long routes such as:

cbosgd!mhuxl!eagle!harpo!seismo!hao!hplabs!hpda!fortune!amd70!decwrl!joe

we type:

joe@decwrl.uucp

This not only saves a lot of typing but allows the machine to pick a "best" route. In this case it might have chosen decvax!decwrl!joe, a route the person may not have known about.

3. Becoming a Domain

For the past 12 years, the ARPANET has used a user@host mailing address syntax. A year ago they realized that a flat addressing structure such as this caused impractical maintenance problems; they then converted to a hierarchical user@domain syntax. The domain is a list of words separated by periods (e.g., F.ISI.ARPA) forming a hierarchy with the top level domain, ARPA, at the right. This method allows a very large number of hosts to be named without any host needing a complete list of all other hosts. Other top level domains, besides ARPA, are possible.

We propose to make UUCP a top level domain. This will enable users on UUCP machines to exchange electronic mail with users of machines in other ARPA domains, such as the ARPANET and CSNET.

In RFC 881*, the ARPANET sets forth a list of requirements for top level domains. These requirements are:

1. There must be a responsible person that can act as coordinator and answer questions. In addition to serving as a central contact point, this person will have the authority to enforce network policies and rules.

Request for Comments 881, Network Information Center, ARPANET.

2. There must be a robust name service. There will be two separate name server machines to which one can send a mail message and receive information back about a particular site.
3. The domain must be of a minimum size. Since UUCP currently has over 2100 hosts, this should be no problem.
4. The domain must be registered with the central domain administrator.

We intend to meet these requirements. In addition, since we will be keeping a list of all host names, it will be possible to ensure that there are no duplications. The UUCP software allows host names of any length, but only uses the first six (System V) or seven (others) characters of the name. We will ensure that all host names are unique in the first six characters and are chosen from a set that will not cause trouble on other machines (lower case letters, digits, and a few punctuation characters such as hyphen and underscore).

The authority issue is slightly sensitive, since there is currently no authority over the UUCP network. Such ultimate authority is, however, necessary to protect other domains and other UUCP machines against unacceptable behavior. Such behavior might involve flooding another machine with large quantities of unwanted mail or generation of traffic that fails to conform to accepted standards and breaks programs on other machines. If a host continues to cause problems after repeated warnings, the site would lose their entry in the data base and their claim to their UUCP host name. We expect use of this authority to be extremely rare. The corresponding authority has existed on the ARPANET for years, yet no site has ever been disconnected because of it.

4. Pathalias

To solve the problem of determining "good" routes to foreign machines, Steven Bellovin has written a program called "pathalias" which he has placed in the public domain. Given a list of sites and their direct neighbors, pathalias computes an "optimal" route from the local host to each other host on the network. Upon receiving a new database, the site administrator runs pathalias on the database and stores the result in a routing file. Any program can then look up the best route to any site in the routing file.

Here is a sample database for a four host network. For each host, the name of the host is given, followed by the names of all sites to which mail can be sent directly. For example, allegra can send mail directly to all three of the other hosts, but gummo can only send mail directly to allegra. In this example, all links are two-way; in the real world, most but not all links are two-way.

```
allegra  cbosgd, gummo, ucbvax
cbosgd   allegra, ucbvax
gummo    allegra
ucbvax   allegra, cbosgd
```

The output from pathalias, run on host cbosgd, is:

```
0       cbosgd                  %s
4000    allegra                 allegra!%s
4000    ucbvax                  ucbvax!%s
8000    gummo                   allegra!gummo!%s
```

The first column indicates the "cost" (this is calculated using defaults); the second column is the machine name; the third column is the mail route to the machine.

Many circumstances can cause routing through a given machine to be more desirable (or less so). Some connections may cost more than others since some links go over leased lines, local area networks, or expensive long distance lines. Some sites may not have an autodialer or may not be able to afford long distance phone calls. Such sites cannot call their neighbors on demand, but must wait to be polled. The frequency of polling varies from every hour to seldom or never, depending on the amount of traffic coming from the other site. Some links are more reliable than others.

Pathalias can find the "lowest cost" path from the local site to each other host on the net if "costs" for connections ("edges") are supplied. Rather than use the default cost of 4000, as above, we can

attach specific costs to each link. In the example below, DEMAND is 300, HOURLY is 500, DAILY/2 is 2500, and DAILY is 5000.

```
allegra  cbosgd(DEMAND), gummo(DAILY/2), ucbvax(HOURLY)
cbosgd   allegra(DEMAND), ucbvax(DEMAND)
gummo    allegra(DAILY/2)
ucbvax   allegra(HOURLY), cbosgd(DAILY)
```

The pathalias computations become:

```
0       cbosgd          %s
300     allegra         allegra!%s
300     ucbvax          ucbvax!%s
2800    gummo           allegra!gummo!%s
```

Changing the "local host" to ucbvax yields:

```
0       ucbvax          %s
500     allegra         allegra!%s
800     cbosgd          allegra!cbosgd!%s
3000    gummo           allegra!gummo!%s
```

The pathalias program can also compute paths to other networks. We believe it is a powerful enough tool to solve the routing problems -- once an accurate database is collected.

5. Data Collection

Currently, there are a few databases scattered throughout the network. Karen Summers-Horton keeps track of those Usenet sites which receive the newsgroup net.announce. Their information includes the site's name, the contact person, electronic and US Mail addresses, and hosts with which the machine exchanges news. Several sites keep track of extensive L.sys files and can call hundreds of machines. These sites have an easy time of finding routes. Rob Kolstad maintains a list of network "edges": connections between pairs of machines. The list is not of ultimate utility since it contains neither connection frequency nor direction. Even though some hosts call others on demand, it is not a good assumption that mail can flow back the other way.

Rob Kolstad (allegra!parsec!kolstad, CONVEX Computer Corp., Dallas), Scott Bradner (allegra!wjh12!sob, Harvard University), and possibly a handful of other volunteers are currently involved in an effort to collect an accurate enough database to generate pathalias style routes for any given machine. Here is a typical entry in their database:

```
=Name:              parsec
=Machine Type/OS:   VAX780/4.1cBSD
=Organization:      PARSEC/Convex Computer Corporation
=Contact Person:    Rob Kolstad
=Electronic-Addr:   parsec!kolstad
=Phone:             214-669-3700
=Postal-Address:    1819 Firman #151, Richardson, TX  75081
=Long/Lat Coors:
=Comments:
=Editor:            parsec!kolstad Tue Jan  2 09:07:00 1984
=
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
#
parsec          unmvax(DAILY/3), rice(DAILY/3), uiucdcs(DAILY/3), ctvax(DEMAND),
                rice(DAILY/3), allegra(DAILY/3), dj3b1(DEMAND), smu(DEMAND)
```

The entries are stored in compressed format (but easily expanded). They have enough information for pathalias to make optimal routes and also include enough site information to solve most problems that come up that require contacting the site.

The database currently contains entries for over 2000 sites. Typically they look like this:

```
=Name:               acsa
=Machine Type/OS:
=Organization:
=Contact Person:
=Electronic-Addr:    acsa!root
=Phone:
=Postal-Address:
=Long/Lat Coors:
=Comments:
=Editor:
=
================================================
#
acsa                 inuxc(DAILY)
```

It is our goal to fill in each of the templates. We currently intend to do this by electronically mailing out our (sometimes partially completed) templates to each machine with directions for completing the form and returning it. We have several scripts and programs to maintain the database and automatically send out mail. We believe the initial data gathering effort will require from 3-9 months.

6. Data Base Maintenance

6.1. Short Term

Short term maintenance of the database will be done strictly by hand. Sites will be encouraged to mail corrections to a central collection point, at which time we will manually update the database. As this will take a considerable amount of time and effort, steps are being taken to automate the process as soon as possible.

6.2. Long Term

In the long term, it is hoped that we can distribute programs (or shell scripts) which will allow simple updating to the database by mailing well-formatted updates to an alias. While this scheme might still require human intervention in the form of "approving" updates, and dealing with improperly formatted requests, it still removes much of the effort from the update process.

We must also develop a "registry" for new sites to consult to verify uniqueness of their site name and enter their initial routing data. While this removes some of the anarchy from the network, it is felt that it is nevertheless a valuable step.

7. Data Base Distribution

Our current database is approximately eight megabytes and growing rapidly, so it is much too large to post to Usenet (even once). However, it is crucial to distribute enough data on a regular basis to allow mail routing programs to work.

One very simple option is to include the database on the USENIX tapes. This option has problems related to timeliness, but an out-of-date database is far more useful than none at all, and it would allow distribution of the entire database.

Another option is to post (on a regular basis) complete information for a small set of backbone sites to Usenet. In addition to this complete information, we would also post a list of all other hosts,

and a path from each host to a backbone site. This should cut the size of our initial distributions by a large factor.

A third option is not to post the entire list of host names. Instead, a hierarchical structure would be worked out in the spirit of ARPANET domains. Thus, an address like cbosgd.uucp might become d.osg.cb.att.uucp, indicating that within the UUCP domain, AT&T region, Columbus location, OSG project, the D machine is specified. No fixed structure such as id.proj.loc.reg.UUCP is implied, each subdomain would subdivide itself as appropriate. The Postal Service and Telephone Company have set up similar hierarchies that can route traffic without any complete lists of all possible locations.

The last two schemes have the disadvantage that they do not explicitly take advantage of the richness of connections that do exist. These problems remain to be solved.

Finally, the name servers to be set up would provide for 'on demand', electronic distribution of directory information for individual sites.

8. Summary

As it stands, the UUCP network is a growing, viable entity which provides very low cost network facilities to well over 2,000 machines. The problems of unreliability, lost messages, and variable propagation delay have been tolerated for many years. Because of the network's growth, its machine-to-machine routing scheme, and lack of centralized control, the requirement for an accurate network-wide map is essential. We need the cooperation of each site in order to construct and maintain a reliable database.

Some Important Addresses

The most important address is that of the EUUG Secretary. From there you can probably get information on anything and anyone, like how to join, who you can turn to for problems, etc.

EUUG Secretary
Owles Hall
Buntingford
Herts SG9 9PL
England

tel: +44 763 71209
UUCP: mcvax!euug

Any articles or material for the Newsletter, except for advertising which is handled by the Secretary above, should go to

Jim McKie
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

tel: +31 20 5924147
UUCP: mcvax!jim

NOTE: unless it comes in machine readable form, it has little chance of being included. What do you have a computer for anyway?

Requests for EUUG Distrubution Tapes should be sent to "EUUG Distributions", also at the above address in Amsterdam.

There are now some National UNIX Systems Users Groups in existence. The addresses of who to contact are

For Denmark (DKUUG)

Keld Simonsen
Indre By-Terminalen
Københavns Universitet
Studiesträde 6
1455 København K
Denmark

tel: +45 1 120115
UUCP: mcvax!diku!keld

For the Netherlands (NLUUG)

Marten van Gelderen
NIKHEF-K
Postbus 411
1009 AJ Amsterdam
The Netherlands

tel: +31 20 5922030
UUCP: mcvax!ikogsmb!csg!marten

For Sweden (EUUG-S)

Bjørn Eriksen
Box 232
S-182 23 Taeby
Sweden

tel: +46 8 7567220
UUCP: mcvax!enea!ber

For the UK (UKUUG)

John Shemeld
The Electronics Lab.
University of Kent
Canterbury CT2 7NT
England

tel: +44 227 66822 x285
UUCP: mcvax!ukc!jds

For France (AFUU)

Association Francaise des Utilisateurs d'UNIX
152 bis Avenue Marx Dormoy
92120 Montrouge
France

tel: +33 1 6554550

For Finland

Timo Kunnas
Helsinki University of Technology
Lab. of Information Processing Science
Otakaari 1
SF-12150 Espoo 15
Finland

tel: +358 4512487
UUCP: mcvax!enea!taycs!jh (Juha Heinaenen)

For Germany
Daniel Karrenberg/Joachim Wolff
Dortmund University
Computer Science Department
Postfach 500 500
D-4600 Dortmund 50
W. Germany

tel: +49 231 7554824
UUCP: mcvax!unido!jw

For Switzerland (UNIGS)
Professor Dr. R. Marty
UNIX Interessengemeinschaft Schweiz
c/o Institut für Informatik
Universitaet Zürich
Sumatrastraße 30
CH-8035 ZäRICH
Switzerland

tel: +41 1 2511872
UUCP: mcvax!appias!circe!marty

EUNET, the European UNIX Network, now spans some 12 European countries with electronic mail and news, and has links to North America. For information on how to hook up, contact

In Continental Europe:

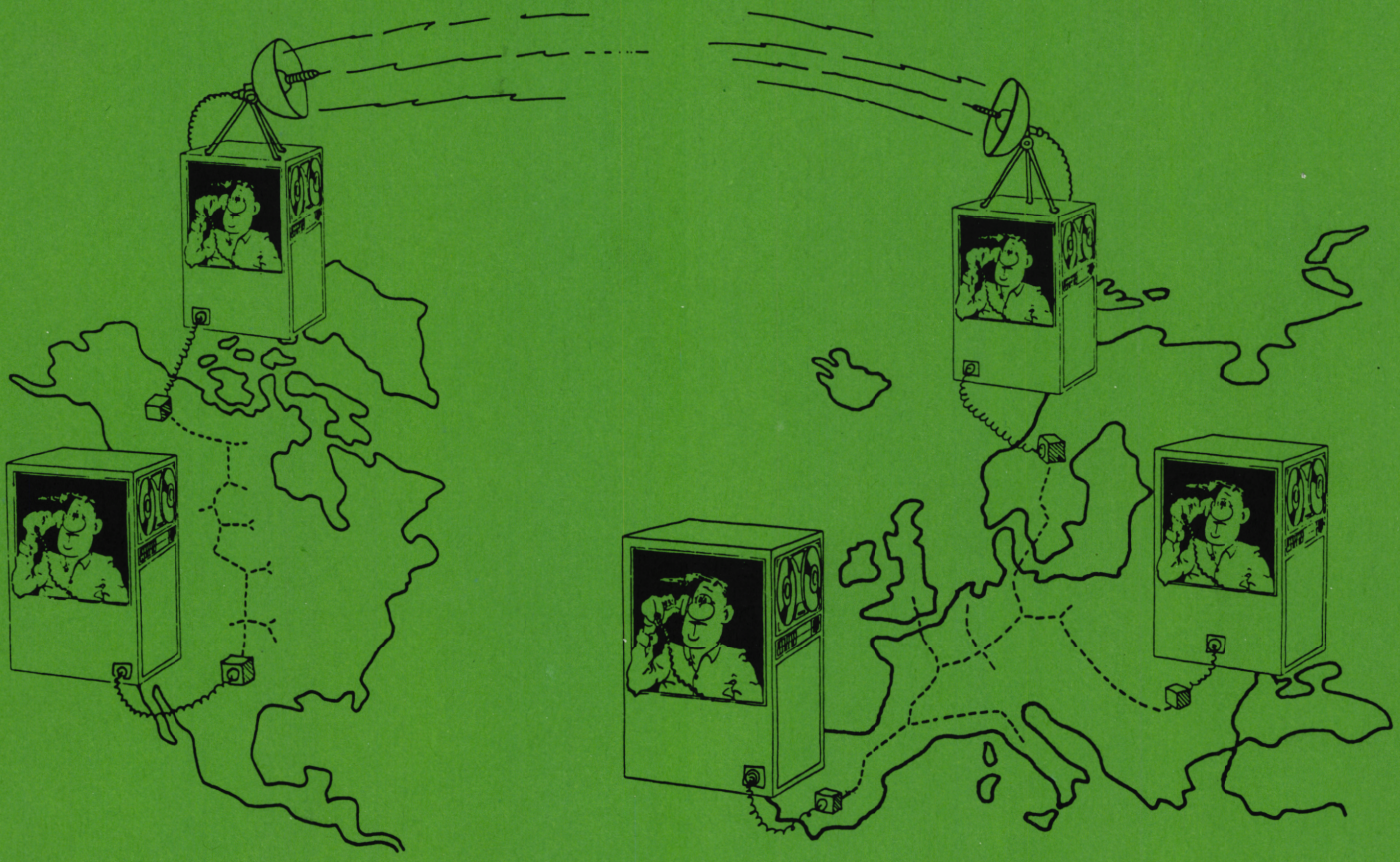
Teus Hagen
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

tel: +31 20 5924127
UUCP: mcvax!teus

In the UK:

Mike Bayliss
Computer Laboratory
University of Kent
Canterbury CT2 7NF
England

tel: +44 227 66822 x7615
UUCP: mcvax!ukc!mjb



The Secretary
European Unix® Systems User Group
Owles Hall
Buntingford, Herts.
SG9 9PL.
Tel: Royston (0763) 73039.