The MODUS Quarterly

Issue # 9

October 1987

Modula-2 News for MODUS, the Modula-2 Users Association

CONTENT

Problems?
Missing a MODUS Quarterly *issue?*
Do you want to subscribe?

Contact your subscription coordinator at one of the addresses given below.

Administration and subscription

USA: Chris Jewell
MODUS
P.O. Box 51778
Palo Alto, CA 94303
E-Mail: chrisj@cup.portal.com
or sun!cup.portal.com!chrisj

Europe: Aline Sigrist
MODUS Secretary
ERDIS SA
P.O. Box 35
CH-1800 Vevey 2
Switzerland

Editor, *MODUS Quarterly*

K. N. King
Department of Mathematics and Computer Science
Georgia State University
University Plaza
Atlanta, GA 30303
(404) 651-2245
E-Mail: king@gatech.edu
BIX: knking

Publisher

Chris Jewell (see above)

Submissions for publication

Send all submissions to the editor. Camera-ready copy is strongly encouraged; however, dot-matrix copy is usually unacceptable. If camera-ready copy cannot be furnished, articles may be submitted on floppy disk (IBM PC only, either 5-1/4" or 3-1/2") or by electronic mail. Articles submitted electronically must not require any subsequent formatting. Files must be either plain ASCII or in Postscript or Microsoft Word format.

The *MODUS Quarterly* welcomes working papers, notes about work in progress, and examples of source code.

Please indicate that publication of your submission is permitted. Correspondence not for publication should be PROMINENTLY so marked.

MODUS

the Modula-2 Users' Association Announces a

General Technical Meeting

When and Where: Aug 18,19; Mt. Hood, OR

The fifth general technical meeting of MODUS (the third in the US) will take place on August 18 and 19, 1988, at the Timberline Lodge on Mt. Hood near Portland, Oregon. All MODUS members and other Modulans are welcome.

The Timberline Lodge is 6,000 feet up an 11,000 foot mountain. You can camp within reasonable driving distance, or stay at the lodge, but there are no other facilities close by. Food at the lodge is good, as is the scenery...

Cost is $76.85 per day (double) or $102.30 (single). These prices include everything: three meals, gratuities, taxes, and the conference room. Please reserve your own room at Timberline (Timberline Lodge, Timberline, Oregon, 97028, 503-226-7979), unless you're coming from overseas and would like assistance (for that, contact Randy Bush at 503-245-2202). The Modus block of rooms (for 40-50 persons) is only reserved until 15 June 1988.

The Timberline Lodge is over an hour from the Portland airport, and ride-sharing is encouraged. Our local host (Randy Bush: see above) has offered to organize and help provide transportation to and from the airport.

Attendance:

If you make a reservation with the Timberline Lodge, you are automatically a meeting attendee; if you plan on camping and attending, please contact Jon Bondy (see below) so we can plan the extra meals (at some minimal cost).

Presentations:

The heart of this meeting is its technical presentations, so **please consider giving some talk** on your area of expertise. Technical papers (to be published in the MODUS News), technical talks, discussion groups, and fora are being solicited. Please contact Jon Bondy, Box 148, Ardmore, PA, 19003, 215-642-1057 (preferably before June 15, 1988) if you wish to give one of the above.

Information for the European-Associated Members

Dear Member

Yesterday-evening I received the originals for Issue #9 from Kim N. King, the new editor of the MODUS Quarterly. Today I will pass it to our printer for the run.

Please remember, the MODUS-Address is

MODUS-Secretary
Aline Sigrist
ERDIS SA
P.O.Box 35
CH-1800 Vevey 2

but the Swiss PTT has changed the phone-numbers; our new phone-number is

021/921 79 80

the phone-number of Heinz Waldburger changed to

021/907 75 75

for international calls, please replace the first "0" by the code for Switzerland.

Please remember, that the MODUS membership is currently only possible for induviduals, not for legal entities. For this reasons we kindly ask you to mention on all correspondance (also payments via the bank!) the name of the member, not the name of the company you are working in. Also the name of your secretary does'nt help so much.

Due to difficulties with the bank (they asked high fees per 45.- Swiss Francs check!) we changed the account to

28'694.01.05
MODUS, Modula-2 Users Association

at

Bank Belp
CH-3123 Belp

Please make bank-transfers to this new bank. For this and other reasons we can not longer accept Checks or other payments in US$. Our poste-office and the printer accepts only Swiss Francs.

By the way: the "Rest of the World" has more than 600 members coming from more than 30 countries.

We wish you a good time and thank you for your support

Aline Sigrist

Vevey, August 19, 1988

Editorial

Change in Editors

You may have noticed a change on the masthead. Dick Karpinski has served as editor of the *MODUS Quarterly* from Issue #0 to the present. Because of Dick's many other activities, he has had less time to devote to the *Quarterly* in recent years, so I've agreed to take the helm. On behalf of all members of MODUS, I'd like to thank Dick for his efforts in getting the *Quarterly* off the ground.

By way of introduction, my name is K. N. (Kim) King. After a number of years as a faculty member at Georgia Tech, I recently moved to Georgia State University, where I'm an associate professor of mathematics and computer science. I'm the author of *Modula-2: A Complete Guide*, recently published by D. C. Heath and Company. I'm a member of both the U.S. and international Modula-2 working groups, and I'm also involved with the Modula-2 Validation Suite.

The *MODUS Quarterly* is the world's leading forum for discussion of Modula-2. With the help of the readership, I look forward to issues filled with useful information and lively debate.

Standardization Update

Because of the *MODUS Quarterly*'s somewhat irregular publication schedule, some readers may not be aware of the status of standardization efforts; here's a quick update.

Efforts to standardize Modula-2 began in England in 1984 with the formation of a working group that is now known as BSI/IST/5/13. In 1986, IST/5/13 asked the International Standards Organization to convene a working group for the purpose of writing an international standard for Modula-2. This group, known as ISO/TC97/SC22/WG13, first met April 1–3, 1987, in Nottingham, England. The second meeting of WG13 was held January 11–15, 1988, in Nice, France. The third meeting is scheduled for August 22–26 in Portland, Oregon.

In addition to the BSI and ISO groups, a number of countries have standardization efforts underway. For the most part, the purpose of these groups is to provide input to BSI and ISO, not to develop separate standards. The U.S. did not have a Modula-2 standards group until late 1987, when the IEEE Microprocessor Standards Committee established working group P1151. P1151 held its first meeting December 17–18, 1987, in Valley Forge, Pennsylvania. The second meeting was held March 21–23 in Atlanta, Georgia. P1151 will meet August 15–17 in Portland, just before the WG13 meeting.

MODUS Meeting

For two weeks in August, Portland will become the center of the Modula-2 universe as both P1151 and WG13 hold meetings. Sandwiched between these two standards meetings will be a two-day MODUS meeting. Don't miss it! Here's a chance to view the beautiful scenery of Mt. Hood, stay in the historic Timberline Lodge, meet the leading luminaries of the Modula-2 world, and find out how standardization is proceeding. See the ad at the front of this issue for complete details. *Note*: Although the ad states that the

block booking at Timberline Lodge expires June 15, it is likely that rooms will be available during the days of the MODUS meeting. Call Timberline to check. Incidentally, Jon Bondy tells me that there is still room for talks and papers. Give Jon a call if you're interested. See pages 21–22 of this issue for a list of the presentations made at last year's MODUS meeting.

About This Issue

Although this issue is dated October 1987, it is being mailed during July 1988. It contains submissions received in late 1986 and early 1987. Be aware that some of the discussion of proposed language changes is now out of date.

One of the most interesting articles in this issue is "Modula-2 Use in Urban Transportation Vital Control," which describes the use of Modula-2 to control the Paris Metro (subway). Show this article to your C and Ada friends who claim that Modula-2 isn't used for anything practical!

Plans for the *MODUS Quarterly*

One of the most important functions of the *MODUS Quarterly* is disseminating information about the continuing efforts to standardize Modula-2. I hope to include more standards papers in future issues.

Another feature I'd like to institute is a list of vendors of Modula-2 compilers, libraries, and related software. It seems to me that the *MODUS Quarterly* is the proper place to collect this kind of information. Note to vendors: Please send me current product literature for this list.

I'd also like to receive articles about the use of Modula-2 in industry. We need to show prospective Modula-2 users that it is just as productive as C and Ada (if not more so) in the "real world."

Your comments, ideas, and (of course) submissions are always welcome. Address them to me at the address shown on the inside front cover or send me an E-mail message.

KNK

Breakthrough™

S O F T W A R E

February 24, 1987

Editor
Modus Quarterly
6521 Raymond Street
Oakland, CA 94609

Dear Richard:

Regarding the suggestion from K John Gough in MQ #6 to have linkers remove from their load images any code not actually referenced. This would reduce the size of the load image, indeed. Nonetheless, it would have an unfortunate side effect.

In our experience with the Logitech M2 system, which we use to develop and support a commercial product (Time Line), we find it valuable to create overlays to the base layer product. Often these are designed and written long after the base layer has been shipped. If the linker removes code that the base doesn't internally reference, then the overlay will not be able to load and execute properly, because base module definitions no longer match their implementations, as loaded.

Respectfully yours,

Andrew Layman

Andrew Layman,
President

Breakthrough Software Corporation ▪ 505 San Marin Drive ▪ Novato, California 94947 ▪ (415) 898-1919

SAVVY COMPUTING, INC.

2 EDGECLIFF ROAD • UPPER MONTCLAIR, NEW JERSEY 07043

March 25, 1987

Barry Cornelius
Department of Computer Science
University of Durham
Durham DH1, 3LE
United Kingdom

Dear Dr. Cornelius:

I am writing, with some trepidation, to express my reservations about the changes being proposed by the Modula-2 Working Group of the British Standards Institute. I realize it is late in the process for objections, however, my concern over the proposals overrides my preference to simply implement the language that others decide upon.

My primary difficulty with the proposed changes is the extent by which existing programs are made invalid. As a user I don't wish to revise existing programs for the sake of compliance, and as a compiler writer, I do not wish to reject programs ported by my clients from other implementations. They are liable to interpret my refusal to compile a previously working program as a bug, instead of a feature. In some cases, I feel that the proposed change detracts from the language, as well as invalidating programs.

Some of the incompatibilities are due to changes that eliminate language features. An example is the removal of the rule for octal constants of the form nnnC (WG086). I agree that CHR(nnn) can be used in its place as suggested, or could if CHR were also still in the language, since CHR, ORD, FLOAT, TRUNC, and VAL are removed by (WG115) and (WG121). I abhor octal, and don't use such constants, but elimination of CHR and ORD will invalidate many of my programs, and those of others as well.

I hope that the M2WG reconsider its decision to restrict HIGH to open array parameters. HIGH can currently be applied to an array without regard to whether it is an argument, private, or local variable. This generality and uniformity of notation is very useful. One should not be required to change every reference to the high value of the index type of an array if the array is changed from a global variable to a procedure argument.

I hope that the M2WG reconsiders the removal of facilities in the language that are considered 'magic' in some sense, such as the optional second argument for INC and DEC (WG122), and the transformation of NEW and DISPOSE into calls to ALLOCATE and DEALLOCATE (WG069 and PIM3). The former weakens INC and DEC so much that there is very little reason to leave them in the language. If we are discarding superfluous features, then INC and DEC are unneeded and should be removed. I would prefer that they be left in the language, and with the optional second parameter, because they are convenient for the programmer, and since they are in the current language.

Removal of NEW and DISPOSE is more serious, since it forces the programmer to specify the sizes of objects being created or destroyed. Current Modula-2 lets the compiler use the type system to ensure that the correct sizes are used. Removal of this valuable facility exposes Modula-2 programmers to a myriad of errors that they need not now cope with. Surely the safety of letting the compiler calculate these sizes is worth a little 'magic'. Compilers are much better at this sort of thing than we are, and this use of the type system is quite in the spirit of Modula-2. It would be safer to reserve ALLOCATE and DEALLOCATE for the cases where explicit specification of the storage size is necessary.

I am also worried by the changes in (WG102) and (WG120), since they use, for safe coercions, what is currently the syntax for unchecked type transfer. Unless I misunderstand the proposal, a statement could be syntactically correct in both the current and the proposed language, but have quite different meanings. This will make it difficult to reliably port programs to the new standard.

I also consider unfortunate the proposal to assign the character SYSTEM.StringTerminator to the rightmost elements of an array variable, when the variable has been assigned from a character string constant shorter than the array (WG106).

I appreciate the desire to cleanly specify the semantics of the assignment by ensuring that the array contents are entirely defined. Nonetheless, other assignments to array variables leave partially defined arrays, and this proposal has the potential for a very high cost in performance. This change could also break existing programs that rely on implementations not altering the positions in the array not overlaid by the string constant (one could argue that they should not have been written that way).

Lest this be seen as special pleading from an implementor, let me point out that the proposed change is very easy to implement on the IBM 370 family, and would require under 20 lines of code added to Modula-2/370. This change would be more expensive on machines lacking block moves with byte propagation. Ironically, if my memory serves me, this would make Modula-2 strings very expensive on the Lilith, which was designed to run Modula-2.

In summary, I believe that any language revision should be as upwardly compatible as possible, and should largely restrict itself to resolving ambiguities and eliminating outright errors in a language.

Non-compatible changes will lead to a proliferation of dialects of Modula-2 that is wider than the relatively small range in existing implementations. This can only lead to confusion that will harm the development of Modula-2 as a portable and widely used language.

I appreciate, respect, and value the work done by the members of the committee, and feel that they have taken steps to what is largely a better language. I hope my comments are taken in this spirit. If my objections are based on misunderstanding the proposed standard, I would be very grateful for any corrections to my comments.

Sincerely yours,

Jeff

Jeffrey Savit

cc: Richard Karpinski, MODUS Quarterly

Thoughts about Modula-2

I am considering M2 as a base for two special-purpose languages (real-time, multi-tasking), although Pascal offers no less to the design. At the same time (concurrently, or else as a variation of the same project), I would like to design a decent native-code optimizing compiler for the Mac. TML has a head start on Pascal (but does not optimize), and I could not beat Apple's own Pascal to the market, so I thought maybe M2 might provide an opportunity to distinguish my work from the masses, so to speak.

At this point it becomes clear that Wirth's '80 ETH report is inadequate as a language specification; you say BSI is involved in a standards effort: I need to get on that mailing list, or least see what they are doing. Perhaps a magazine focussing on M2 would give some insight on problem areas to watch for.

Some of my own thoughts, based on the Report:

* Sets that do not allow a base type of character are inadequate or even useless.

* I would like some way for the user to specify different word sizes for variables, so to take advantage of speed and space costs in the target machine. I could have the compiler data-flow analysis infer some of this, but hints from the programmer are indispensible. At the very least, I could base it on subranges of some _huge_ integer type.

* The Mac has two different parameter-passing protocols for procedure calls; I need some way to inform the compiler which to use. Ideally I would like to specify all ROM calls as if they were M2 calls. Is there a place for directives like Pascal's 'forward', where additional directives such as 'inline', 'trap=A970H', etc., could communicate this information to the compiler? Would also like to be able to inform the compiler about kinds of optimizations to invoke.

* One of the things that makes developers take notice is the turn-around time from edit to test, that is, the compile-link throughput. What I would like to do is facilitate this by giving the compiler a debug-generate mode that is _fast_, including all linking of precompiled modules (maybe no optimization). I'm thinking about compiling directly to code in this mode, and linking other modules in at compile-time, probably as separate code segments. Not sure how to fit this into the M2 schema, but it seems appropriate.

* It is not at all clear in the Report how to make interrupts work, but I suppose I could come up with something workable. I'd hate to devise something different than the rest of the M2 world.

* Did I understand you to say type-casting is no longer a global facility (I hope that is the case)? It should be restricted to modules that are deliberately doing strange stuff. It seems from the Report that I might be able to restrict it to modules that import from SYSTEM, but there must be a better, more explicit, way.

* I resent the requirement that reserved words be CAPITALIZED, since that makes typing clumsy. If M2 does not have a conventional way to desensitize the compiler to case, I will invent one (as an option, of course, perhaps in the Mac menu). A preprocess filter program is

not an acceptable solution (see turn-around time, above).

Development of such a compiler is likely to be from scratch, so that I have full control of compile time and intermediate code format. However, I could probably make better progress if I had working M2 tools to develop on. I might be willing to build on an existing parser, if it were available to me and if it were adaptable. My experience, however, is that adapting old programs to make new is not practical, even if I wrote the old. I presently have access to a Sage 2 (one potential client language must use Sage/Pinnacle as host), but I do all new work on Mac since it is _so_ much better an environment that anything else I ever used. The other client language will probably be constrained to run on IBM-PC host (but I won't feel too badly if compile-time turnaround is not as good as on the Mac).

Ultimately I hope to embed the compiler in a Mac-Application generator, but not immediately (one step at a time).

[a second e-mail note:]

It seems to me that M2 has some ugliness that makes clean compilers hard to write -- e.g. nested comments that cannot be filtered out by a finite state machine scanner, so you need a PDA between the scanner and parser -- and some ugliness that makes it hard to use -- like the requirement of reserved words all caps, which focusses your attention on low-level structure instead of high level content. It appears that (like C) M2 was designed with the PDP-11 in mind, and it shows.

My current thinking is to extend the language is some of the following ways: (optional) case-desensitizer, Pascalish read and write that compile to low-level InOut calls (a la New -> ALLOCATE), and allow comparison of structures (particularly strings) with '>', '<', etc. The Program module is almost impossible to implement correctly in a Mac and have it do reasonable things; I see why nobody tried a M2 on the Mac. The definition of strings is clumsy (you get the terminator only *sometimes*); to make it work on the Mac I have to implement both M2 strings and length-based strings transparently.

I still don't see how to implement reasonable concurrency without burdening the concurrent processes with a lot of low-level details best left in the operating system. Sorry, this was not intended to turn into a gripe session, but if I'm going to *sell* compilers, I have to have reasonable solutions to the problems.

[and a third e-mail:]

It is interesting that Sale did not realize --and it is not clear that Pattis noticed-- that in Sale's original program (never mind the bug) padding out with nulls has no effect whatsoever; a single null terminator gives the same results. Thanks for forwarding the material: it was interesting reading.

BTW, I cannot find anything in M2 that gives the functionality of Pascal's "Packed" attribute. How are we to force dense implementations of records and arrays, such as with files and hardware registers?

I finished Gleaves' book, and am beginning to get confused about what is _in_ Modula-2 and what is not. Anything current?

The Problem of Nested Comments

With the innovation (over Pascal) of nested comments, Modula-2 has raised some interesting ambiguities and one (slightly) annoying problem for compiler designers.

Pascal was nice from a compiler construction point of view, because you could have a pure Finite State Machine (FSM) to do lexical analysis and an almost pure push-down automaton (PDA) to do the parsing, stumbling only over the optional "else". Modula-2 has cleared up the "else" problem, only to introduce a somewhat stickier lexical problem in the syntax of comments.

The problem is that you cannot count comment delimiters in a FSM. It requires a PDA. You cannot just insert the comment syntax into the parser grammer, because the fact that comments are valid anywhere blanks are makes a single grammar encorporating comments horribly complex. Nobody really wants to interpose a second PDA between the parser and the scanner (lexical analyser) for efficiency reasons. The only reasonable alternatives seem to be

1) a hack that counts comment delimiters between the parser and the scanner, or
2) a hacked-up state in the scanner that counts delimiters.

Putting the hack inside the scanner means that you cannot comment out statements with string constants containing comment delimiters '(*' or '*)'. Putting the hack outside the scanner means that you cannot allow apostrophes in comments except in balanced pairs. There is a certain interaction between comments and string constants. For example, which of the following lines represent valid Modula-2 code?

```
Stringl = '*)';                                  (* obviously ok *)
(* Stringl = '*)'; *)                        (* not so obvious! *)
(* there isn't anything wrong with this line, or is there? *)
(* this isn't better *) (* isn't it? *) (* 1 comment or 2? *)
(* Stringl = '*) This is almost wierd enough to fail (*';  *)
```

I have not tried any of these lines on a Modula-2 compiler to see what they do, but I'd bet that not all compilers work the same way. Note that in Pascal the question never came up, since the first occurrence of a end-comment delimiter was the end of the comment; a FSM could be used and that defined the meaning of quoted strings within comments (i.e. no significance). Not all statements could be "commented out" so nobody questioned that the pathological cases above failed.

The question can be resolved either by disallowing apostrophes in comments except as balanced quotes, or by disallowing the practice ofyl "commenting out" statements -- at least those containing quoted strings that include comment delimiters. The first is surely an offence to the proper grammarians among us; the second eliminates the only plausible justification for allowing nested comments at all. In any case it should be specified clearly in any Modula-2 standard.

Tom Pittman
Dept. Computer Science
Kansas State University
Manhattan, KS 66506

Christian Tanzer
Glasauergasse 32
A-1130 Vienna, Austria
Phone: (222) 824-8764

July 27, 1987

Dr. Richard Karpinski
MODUS Quarterly editor
6521 Raymond Street
Oakland, CA 94609

Dear Dr. Karpinski,

after looking over all the MODUS Quarterly's, I'd like to express my views about some points I find important.

Before I learned about *Modula-2*, I used mainly FORTRAN as programming language. Since spring 1986 I program in *Modula-2* primarily. Currently I am working on a program for control of an astronomical photometer. The computer used is a IBM-compatible, the *Modula-2*-implementation is Logitech's. Although there is a lot of hardware programming necessary and I had to implement a number of interrupt service routines, only one module required the use of Assembler language (for the implementation of 32- and 64-bit CARDINAL arithmetic). *Modula-2* has proved to be ideally suited for hardware programming.

To have the Modulans see some code in MODUS Quarterly, I've enclosed the listings of a module for string handling, which was my first exercise in the usage of *Modula-2*.

I like the style and the contents of MODUS Quarterly very much, but some articles in the early issues were copied so badly that I couldn't read a word of them.

I've used Logitech's *Modula-2* for the IBM-PC, MacMETH on the Macintosh, and the Hamburg implementation of *Modula-2* for VAX/VMS.

Yours sincerely

Christian Tanzer

Christian Tanzer

Views of a programmer

I think that CARDINAL should NOT be defined as subrange of INTEGER or be removed from the language. I use CARDINAL's more frequently than INTEGER's and I often need the whole range offered by CARDINAL. I believe that the availability of unsigned arithmetic is a definite advantage of MODULA-2. Moreover the revision of the MODULA-2 standard by Niklaus Wirth states that the type ADDRESS has to be compatible to either CARDINAL or LONGCARD, which is not possible, if CARDINAL is a subrange of INTEGER.

I'd like to have the possibility to define large sets like SET OF CHAR. It is very convenient to define constant SETs OF CHAR in many situations.

I think that there should NOT be included a new type STRING as proposed by the BSI standard group. I like the proposal to patch up the existing definition as described in MODUS Quarterly #6, page 19, much better.

I wonder why MODULA-2 does not allow CONSTANT's of structured type. It is a waste of run-time and program space to define VAR's and to initialize them in the module body or at the begin of a procedure, if you just need a constant. Moreover the export of a variable, which is meant to be a constant, is very dangerous.

I think that the BSI proposal concerning type conversions is much better than the solution taken by the ETH single pass compiler, which uses VAL for unsafe type conversions and disallows the use of type-identifiers for conversions.

Concerning constant expressions, I think that any expression should be allowed which can be evaluated by the compiler. That rules out functions like MathLib0.sin.

I'd like to have the possibility to define POINTER TO ARRAY OF type. In an interactive application for data reduction you do not know at programming time, how large the data arrays of the user will be. I do not believe that it is a clean solution to define

```
TYPE
    largeArray:     ARRAY [0..largestExpectedSize-1] OF type;
VAR
    a:              POINTER TO largeArray;
```

as I'm currently forced to do. Even worse seems to me the possibility to use an ADDRESS instead of a pointer.

On machines which are able to address single bytes, ARRAY OF BYTE should be compatible to all types, because ARRAY OF WORD can be compatible only to types which size is a multiple of words.

A definition module should export all objects automatically. The EXPORT statement (in definition modules) should be dropped from the standard.

I'd like to have module Storage export a procedure Available. NEW and DISPOSE should be removed from the language (as was done by the ETH single pass compiler).

I'd like to have some module (e.g., Coroutines) export procedures to change the current priority of a process. Often only a small part of a procedure requires a different priority, but factoring out that part into another procedure exported by a different module would give a bad design. I agree with the proposal that Coroutines exports a type PRIORITY as is described in MODUS Quarterly #8, page 54.

I strongly believe that is is necessary to provide a possibility to call non-MODULA-2 procedures from a MODULA-2 program. Wirth's statement "rewrite existing libraries, you will make them better" is just ridiculous.

- who would try to rewrite all library procedures available in VMS?
- who can afford to rewrite libraries like NAGLIB or CERNLIB?
- who is going to implement his own version of GKS in MODULA-2?
- if you have an existing interactive data analysis system written in FORTRAN and you want to change just the user interface, should you
  - write it again in FORTRAN?
  - rewrite all data analysis procedures which work perfectly well and are really good debugged after years of usage by thousands of users?
  - patch your MODULA-2-compiler?

I think that the possibility to define external procedures should be content of a MODULA-2 standard.

I heartily agree with the BSI standard group that many points concerning coroutines have to be changed or clarified. I think that every implementation of MODULA-2 should give some hints about the size of workspace needed by a coroutine. Where does the uninformed programmer get the informations necessary to make an "informed guess"?

I think that dynamic linking is very useful if it is implemented efficiently and should be provided by a sound MODULA-2 implementation.

```
(**
* DEFINITION MODULE STRINGS
*
* Purpose
*    Provide string handling routines
*
* Comment
*    A constant string is a sequence of characters enclosed in either double
*    quotes or apostrophes.
*    A (variable) string is an array of CHAR's. If the string is shorter
*    than the length of the array it is terminated by 0C.
*    The maximum length of a string is given by HIGH (string-variable),
*    the actual length of the string is returned by the procedure
*    Length (string-variable).
*
*    For a string s of length l the first character is contained in
*    s [0], the last character is contained in s [l-1]; if l <= HIGH (s)
*    then s [l] = 0C; if l = HIGH (s) + 1 then no 0C is contained in s
*    and the last character is contained in s [HIGH (s)].
*
*    For substrings the following relations hold:
*        If a is the position of the first character of the substring of
*        length l then the position of the last character is
*
*            o = a + (l - 1)    -->     l = (o - a) + 1
*
*    For all string handling procedures the following convention applies:
*        If the length of the result of a string operation exceeds the
*        maximum length of the destination string then it is truncated
*        to the length of the destination string (no error is signaled).
*
*    If a find-operation does not find the search string then
*    Length (source-string) is returned as result.
*
* Author
*    Christian TANZER
*
* Revision dates
*    14-APR-86 -- Creation
**)
```

```
DEFINITION MODULE STRINGS;

   FROM CHRSET IMPORT CHARSET;

   EXPORT QUALIFIED
      LT,       EQ,       GT,                                        (* constants *)
      CompVal,                                                       (* type *)
      Index,    Compare,  Length,  Assign,                           (* procedures *)
      XtractP,  XtractL,  DeleteP, DeleteL, Insert, Replace,         (*    .       *)
      Trim,     Concat,   Upcase,  Find,    Skip,   Remove;          (*    .       *)

    CONST
        LT = -1; EQ = 0; GT = 1;     (* less than; equal; greater than *)
    TYPE
        CompVal = [LT .. GT];        (* result of string comparison *)

    PROCEDURE Index   (String: ARRAY OF CHAR; Substr: ARRAY OF CHAR;
                       Apos: CARDINAL): CARDINAL;
                      (* Returns the start position of Substr in String,
                         the search operation starts at position Apos;
                         IF Substr is not contained in String THEN
                             Index := Length (String) *)

     PROCEDURE Compare (String1, String2: ARRAY OF CHAR): CompVal;
                      (* If the strings are unequal in length, the
                         shorter string is considered to be filled with blanks
                         to the length of the longer string *)

      PROCEDURE Length  (String: ARRAY OF CHAR): CARDINAL;
                       (* Length is the position of the first 0C or
                          (HIGH (String) + 1) if no 0C is contained in String *)

      PROCEDURE Assign  (Src: ARRAY OF CHAR; VAR Dst: ARRAY OF CHAR);
                       (* Dst := Src *)


      PROCEDURE XtractL (Src: ARRAY OF CHAR; Apos, Len: CARDINAL;
                         VAR Dst: ARRAY OF CHAR);
                       (* Copy Len characters of Src to Dst, first
                          character copied is Src [Apos] *)

     PROCEDURE XtractP (Src: ARRAY OF CHAR; Apos, Opos: CARDINAL;
                         VAR Dst: ARRAY OF CHAR);
                       (* Copy substring of Src to Dst, first
                          character copied is Src [Apos], last
                          character copied is Src [Opos] *)

     PROCEDURE Replace (Src: ARRAY OF CHAR; Apos: CARDINAL;
                         VAR Dst: ARRAY OF CHAR);
                       (* replace substring of Dst - starting at position
                          Apos - by the string Src *)

     PROCEDURE Insert  (Src: ARRAY OF CHAR; Apos: CARDINAL;
                         VAR Dst: ARRAY OF CHAR);
                       (* insert Src into Dst starting at position Apos *)

     PROCEDURE DeleteP (Apos, Opos: CARDINAL; VAR Dst: ARRAY OF CHAR);
                       (* Delete the characters Dst [Apos] to Dst [Opos] *)

     PROCEDURE DeleteL (Apos, Len: CARDINAL; VAR Dst: ARRAY OF CHAR);
                       (* Delete Len characters of Dst starting with
                          Dst [Apos] *)
```

```
PROCEDURE Trim     (VAR String: ARRAY OF CHAR): CARDINAL;
                   (* Remove trailing blanks from String; the length
                      of the Trimmed string is returned *)

PROCEDURE Concat   (Src1, Src2: ARRAY OF CHAR; VAR Dst: ARRAY OF CHAR);
                   (* append source2 to source1 and copy it to Dst
                      Dst := Src1 // Src2 in Fortran 77 syntax *)

PROCEDURE Upcase   (VAR String: ARRAY OF CHAR);
                   (* Convert String to uppercae *)

PROCEDURE Find     (SearchChars: CHARSET; Apos: CARDINAL;
                     String: ARRAY OF CHAR): CARDINAL;
                   (* Find position of next character of string which is
                      element of SearchChars, beginning the search
                      with String [Apos];
                      if such a character is not contained in String then
                      Find := Length (String) *)

PROCEDURE Skip     (SkipChars: CHARSET; Apos: CARDINAL;
                    String: ARRAY OF CHAR): CARDINAL;
                   (* Find position of next character of string which is
                      not element of SkipChars, beginning the search
                      with String [Apos];
                      if such a character is not contained in String then
                      Skip := Length (String) *)

PROCEDURE Remove   (SearchChars: CHARSET; VAR String: ARRAY OF CHAR;
                    VAR Length: CARDINAL);
                   (* Remove all characters which are
                      element of SearchChars from String;
                      Length is the number of remaining characters *)

END STRINGS.
```

```
(**
* IMPLEMENTATION MODULE STRINGS
*
* Purpose
*    Provide string handling routines
*
* Author
*    Christian TANZER
*
* Revision dates
*    11-APR-86 -- Creation
**)
IMPLEMENTATION MODULE STRINGS;

    FROM CHRSET IMPORT CHARSET, FullChSet, In, Incl, Excl;

    CONST
        TAB = 11C;

    VAR
        NotBlankSet, LcLetSet: CHARSET;
        ch: CHAR;

     PROCEDURE MIN (i, j: CARDINAL): CARDINAL;
     BEGIN (* procedure MIN *)
         IF i < j THEN RETURN i; ELSE RETURN j; END;
     END MIN;

     PROCEDURE Index (String, Substr: ARRAY OF CHAR;
                      Apos: CARDINAL): CARDINAL;
         VAR
             i, j, k, l1, l2: CARDINAL;
             ld:              INTEGER;
             found:           BOOLEAN;
     BEGIN
         l1 := Length (String);   l2 := Length (Substr);
         ld := INTEGER (l1) - INTEGER (l2);
         i  := Apos; found := FALSE;
         LOOP
             IF found OR (INTEGER (i) > ld) THEN EXIT END;
             j := 0; k := i;
             LOOP
                 IF    j >= l2  THEN found := TRUE; EXIT;
                 ELSIF String [k] # Substr [j] THEN EXIT END;
                 INC (j); INC (k);
             END;
             INC (i);
         END;
         IF NOT found THEN RETURN l1;
                      ELSE RETURN i - 1;
         END;
     END Index;
```

```
PROCEDURE Compare (String1, String2: ARRAY OF CHAR): CompVal;

    VAR l1, l2: CARDINAL;

    PROCEDURE Comp (s1, s2: ARRAY OF CHAR;
                                    ll, ls: CARDINAL): CompVal;
        VAR i: CARDINAL;
    BEGIN (* procedure Comp *)
        FOR i := 0 TO ls - 1 DO
            IF s1 [i] < s2 [i] THEN RETURN LT; END;
            IF s1 [i] > s2 [i] THEN RETURN GT; END;
        END;
        FOR i := ls TO ll - 1 DO
            IF s1 [i] < ' ' THEN RETURN LT; END;
            IF s1 [i] > ' ' THEN RETURN GT; END;
        END;
        RETURN EQ;
    END Comp;

BEGIN (* Compare *)
    ll := Length (String1); l2 := Length (String2);
    IF ll >= l2 THEN RETURN  Comp (String1, String2, ll, l2);
                ELSE RETURN -Comp (String2, String1, l2, ll); END;
END Compare;

PROCEDURE Length (String: ARRAY OF CHAR): CARDINAL;
    VAR i, l: CARDINAL;
BEGIN
    i := 0; l := HIGH (String);
    LOOP
        IF (i > l) OR (String [i] = 0C) THEN EXIT END;
        INC (i);
    END;
    RETURN i;
END Length;

PROCEDURE Assign (Src: ARRAY OF CHAR;
                  VAR Dst: ARRAY OF CHAR);
    VAR i, l, ls, ld: CARDINAL;
BEGIN
    ls := Length (Src); ld := HIGH (Dst) + 1; l := MIN (ls, ld);
    FOR i := 0 TO l - 1 DO Dst [i] := Src [i]; END;
    IF l < ld THEN Dst [l] := 0C; END;
END Assign;
```

```
PROCEDURE XtractP (Src: ARRAY OF CHAR;
                   Apos, Opos: CARDINAL;
                   VAR Dst: ARRAY OF CHAR);
BEGIN (* procedure XtractP *)
    IF Opos >= Apos THEN
        XtractL (Src, Apos, Opos - Apos + 1, Dst);
    ELSE
        Dst [0] := 0C;
    END;
END XtractP;

PROCEDURE XtractL (Src: ARRAY OF CHAR;
                   Apos, Len: CARDINAL;
                   VAR Dst: ARRAY OF CHAR);

    VAR i, j, len, ls, ld, opos: CARDINAL;

BEGIN (* procedure XtractL *)
    ls  := Length (Src);  ld   := HIGH (Dst) + 1;
    len := MIN (Len, ld); opos := MIN  (ls, Apos+len) - 1;

    i := 0; FOR j := Apos TO opos DO Dst [i] := Src [j]; INC (i); END;
    IF i < ld THEN Dst [i] := 0C; END;
END XtractL;

PROCEDURE DeleteP (Apos, Opos: CARDINAL; VAR String: ARRAY OF CHAR);
BEGIN (* procedure DeleteP *)
    IF Opos >= Apos THEN DeleteL (Apos, Opos - Apos + 1, String); END;
END DeleteP;

PROCEDURE DeleteL (Apos, Len: CARDINAL;  VAR String: ARRAY OF CHAR);

    VAR i, j, l: CARDINAL;

BEGIN (* procedure DeleteL *)
    l := Length (String); j:= Apos;
    FOR i := Apos+Len TO l-1 DO String [j] := String [i]; INC (j); END;
    IF j < l THEN String [j] := 0C; END;
END DeleteL;

PROCEDURE Insert (Src: ARRAY OF CHAR; Apos: CARDINAL;
                  VAR Dst: ARRAY OF CHAR);

    VAR i, j, ls, ld, lm: CARDINAL;

BEGIN (* procedure Insert *)
    ls := Length (Src); lm := HIGH (Dst) + 1;
    ld := MIN (ld, ls + Length (Dst));

    j := Apos;
    FOR i := Apos + ls TO ld - 1 DO Dst [i] := Dst [j]; INC (j); END;
    IF ld < lm THEN Dst [ld] := 0C; END;
    Replace (Src, Apos, Dst);
END Insert;
```

```
PROCEDURE Replace (Src: ARRAY OF CHAR; Apos: CARDINAL;
                   VAR Dst: ARRAY OF CHAR);

    VAR i, j, ls, ld, opos: CARDINAL;

BEGIN (* procedure Replace *)
    ls := Length (Src); ld := Length (Dst);
    opos := Apos + (ls - 1); opos := MIN (opos, HIGH (Dst));

    i := 0; FOR j := Apos TO opos DO Dst [j] := Src [i]; INC (i); END;
    IF (opos >= ld) & (opos < HIGH (Dst)) THEN Dst [opos+1] := 0C; END;
END Replace;

PROCEDURE Trim (VAR String: ARRAY OF CHAR): CARDINAL;

    VAR l: CARDINAL;

BEGIN
    l := Length (String);
    IF l > 0 THEN
        LOOP
            DEC (l);
            IF In (String [l], NotBlankSet) THEN INC (l); EXIT; END;
            IF l = 0 THEN EXIT END;
        END;
    END;
    String [l] := 0C; RETURN (l);
END Trim;

PROCEDURE Concat (Src1, Src2: ARRAY OF CHAR;
                  VAR Dst: ARRAY OF CHAR);

    VAR i, j, ls1, ls2, ld: CARDINAL;

BEGIN
    ls1 := Length (Src1); ls2 := Length (Src2); ld := HIGH (Dst) + 1;
    ls1 := MIN (ls1, ld); ls2 := MIN (ls2, ld-ls1);

    j := ls1;
    FOR i := 0 TO ls1 - 1 DO Dst [i] := Src1 [i]; END;
    FOR i := 0 TO ls2 - 1 DO Dst [j] := Src2 [i]; INC (j); END;

    IF j < ld THEN Dst [j] := 0C END;
END Concat;

PROCEDURE Upcase (VAR String: ARRAY OF CHAR);

    VAR i, l: CARDINAL;

BEGIN
    l := Length (String);
    FOR i := 0 TO l - 1 DO
        IF In (String [i], LcLetSet)
            THEN DEC (String [i], ORD ('a') - ORD ('A')) END;
    END;
END Upcase;
```

```
PROCEDURE Find (SearchChars: CHARSET; Apos: CARDINAL;
                String: ARRAY OF CHAR): CARDINAL;

    VAR i, l: CARDINAL;

BEGIN
    l := Length (String); i := Apos;
    WHILE (i < l) & (NOT (In (String [i], SearchChars))) DO INC (i); END;
    RETURN i;
END Find;

PROCEDURE Skip (SkipChars: CHARSET; Apos: CARDINAL;
                String: ARRAY OF CHAR): CARDINAL;

    VAR i, l: CARDINAL;

BEGIN
    l := Length (String); i := Apos;
    WHILE (i < l) & (In (String [i], SkipChars)) DO INC (i); END;
    RETURN i;
END Skip;

PROCEDURE Remove (SearchChars: CHARSET;
                  VAR String: ARRAY OF CHAR; VAR Ld: CARDINAL);

    VAR i, l: CARDINAL;

BEGIN
    l := Length (String); Ld := 0;
    FOR i := 0 TO l DO
        IF (NOT (In (String [i], SearchChars))) THEN
            String [Ld] := String [i]; INC (Ld);
        END;
    END;
    String [Ld] := 0C;
END Remove;

BEGIN (* STRINGS *)
    NotBlankSet := FullChSet; Excl (NotBlankSet, ' '); Excl (NotBlankSet, TAB);

    FOR ch := 'a' TO 'z' DO Incl (LcLetSet, ch); END;
END STRINGS.
```

MODUS Conference June 15 & 16, 1987

Seven Hills Conference Center, San Francisco, California.

Monday, June 15, 1987

Introduction, George Symons, Stan Osborne
Standardization Panel, Moderator: Dick Karpinski
A discussion of Opaque Types, Chuck Bilbe
Testing Floating Point Implementations, Dick Karpinski
Converting Big UCSD Pascal Applications to M2, Jon Bondy
Writing Portable Applications,Morris Djavaheri
Building Large Applications and Support Tools, Ed de Young
Adding High Level Concurrency Features, Mike Meehan

Tuesday, June 16, 1987

Modula-2 for NS32000 Embedded Systems, Peter Ashenden
Modula-2/370, Experiences with M2 as a portable Systems Language, Jeffrey Savit
ISO WG13 Proposed Concurrent Programming Libraries, Stan Osborne
Design Language From Modula-2, "MODEL", Frode Odegard
Form-maker, a screen/form generator for M2, David Rhoads
Building a Network Simulator, Paul Labbe'
Using the ISO Networking model for communication, Andy Bierman

Technical Demos

Andy Bierman, Two IBM PC/AT's communicating by serial cable.

Workman & Associates, IBM PC & Atari ST demos.

Abstracts

High Level Language Concurrency Features for Distributed Computation

Mike Meehan, The University of Alabama in Huntsville.

The analysis and formulation of concurrent programming constructs suitable for implementation in high level languages targeted for distributed programming environments is discussed. Concurrent programming abstractions in common use in high level languages are analyzed. An alternative formulation called the dynamic monitor is given. The dynamic monitor concept is demonstrated through incorporation into a high level programming language, M3. M3 is derived from Modula-2. M3 is a superset of Modula-2 designed for distributed computation in loosely coupled network environments. A compiler, for this language, generating instructions for a local area network of micro-computers is presented.

Modula-2 for NS32000 Embedded Systems

Peter Ashenden, Department of Computer Science, University of Adelaide, Australia.

The development of a cross-support system for the Modula-2 language is discussed. The cross-compiler runs under the VAX/VMS operating system, generating code for the NS32000 processor; it was used in the development of embedded software for the QDS-1000 workstation. Code generated by the Modula-2 cross-compiler can be run either stand-alone or under the EXEC run-time executive supplied by National Semiconductor. The development of the latter version provided some examples of situations where EXEC strongly supported the kind of run-time environment required for Modula-2, as well as giving rise to some difficulties because of interference between EXEC and the Modula-2 run-time organization.

Modula-2/370, Experiences with Modula-2 as a portable systems language.

Jeffrey Savit, Savvy Computing, Inc., Upper Montclair, New Jersey.

A discussion of the Modula-2/370 implementation and the decisions made during development. Also discussed are aspects of the language and common programming practices that assist or impede portability.

MODEL - a Modula-2 Design Language

Frode Odegard, Modula-2 CASE Systems A/S, Jessheim, Norway.

Introduces MODEL, a notation which lets designers divide projects into subsystems. Each subsystem consists of modules. The language lets you make modules private to their subsystem, prevent unwanted dependencies and set up forced dependencies on the module level. The main point is to help designers enforce further rule upon the implementors, other than those offered by Modula-2. Finally, a MODEL-based database used in the PEM system is described.

Building a Network Simulator

Paul Labbe', Communications Research Centre, Ottawa, Ontario, Canada.

Building a network simulator for mobile platforms provides a valuable experimental environment for the control and observation of an event-driven simulation. To cope with the evolution of the communications systems being simulated a simulator was needed that could be easily modified. The use of abstract data types, information hiding, and separate compilation allows building the components of the simulator concurrently. Tools were developed to ease building and testing the simulator. These tools and the Modula-2 compiler are used to guarantee the consistency of the simulator during its development.

Problems with the Definitions of ORD and VAL

Issue 2: 1st September 1987

Barry Cornelius

Computer Science Subject Group
School of Engineering and Applied Science
University of Durham
Durham DH1 3LE England

1. Introduction

The Modula-2 Report defines ORD(x) by:

> ordinal number (of type CARDINAL) of x in the set of values defined by type T of x. T is any enumeration type, CHAR, INTEGER, or CARDINAL.

and defines VAL(T,x) by:

> the value with ordinal number x and with type T. T is any enumeration type, or CHAR, INTEGER, or CARDINAL. VAL(T,ORD(x))=x, if x of type T.

There are difficulties with these definitions when T is a subrange type or is the type INTEGER. It is the aim of this paper to explain these difficulties.

2. Terminology

The terms "whole-number-type" and "exception" are used in this paper.

> By a "whole-number-type" we mean an integer or cardinal type. (We speak of "a" cardinal type in case there is more than one.) By an "exception" we mean a run-time event beyond which the semantics of the program is undefined. (Implementations may produce a warning on an exception, provide some non-standard recovery or just continue processing.)

The above definitions are taken from the paper "Type Conversions in Modula-2" by Brian Wichmann ("MODUS Quarterly" Issue 6, pp. 21-24).

3. Use of ORD with Subrange types

Given:

```
TYPE
   months=[1..12];
   lengths=[28..31];
VAR
   m:months;
   n:lengths;
...
m:= 1;
n:= 28;
```

what is the value of ORD(n)?

Some have argued that ORD(n) should deliver 0 because the value of n is the first value in the set of values of the type of n. However, this interpretation would cause difficulties for calls like ORD(n-1) and ORD(n+m). It would also mean that the ordinal number of a value changes depending on the subrange that is chosen.

Instead the value of ORD(n) is 28. This is because of the following reasons. In general, the parameter to ORD is an expression. Now, any operand in an expression which is a variable of a subrange type is treated as if it were of the host type of the subrange type. Thus, the n in ORD(n) is considered to be of type CARDINAL and so ORD(n) has the value 28. If this approach is adopted then all the problems disappear. Pascal does it in this way --- section 6.7.1 of the ISO Pascal Standard says:

> Any factor whose type is S, where S is a subrange of T, shall be treated as of type T.

4. Use of VAL with Subrange types

Given the types:

```
TYPE
   day=(sun, mon, tue, wed, thu, fri, sat);
   work=[mon..fri];
```

then there is probably no disagreement that VAL(day,1) has the value mon. But how about VAL(work,1)?

It might be argued that VAL ought to be illegal when T is a subrange type because Wirth's definition of VAL states that "T is any enumeration type, or CHAR, INTEGER, or CARDINAL" and thus subrange types are not included. However, I guess most people would argue that this was not intended.

Although some would argue that VAL(work,1) ought to have the value tue, I believe that VAL(work,1) also has the value mon. Recall that

VAL(T,x) is defined as "the value with ordinal number x and with type T". Now, the ordinal number of the value mon is 1 and mon is also a value of the type work. Hence, it satisfies the definition. Note that the condition VAL(T,ORD(x))=x also holds.

From this, it follows that VAL(work,0) should lead to an exception since there is no value of the type work that has ordinal number 0.

5. Use of ORD and VAL with the type INTEGER

What is the value of ORD(-1)? There seems to be (at least) four possible answers:

(a) -MIN(INTEGER)-1

(b) -1

(c) 1

(d) an exception

I'll look at each of these in turn.

5.1 ORD(-1) has the value -MIN(INTEGER)-1

The Modula-2 Report states that ORD delivers a value of type CARDINAL. The way to map all the INTEGER values onto CARDINAL values is as follows:

```
ORD(-32768) =     0
...
ORD(    -1) = 32767
ORD(     0) = 32768
ORD(     1) = 32769
...
ORD( 32767) = 65535
```

Here I have assumed particular values for MIN(INTEGER) and MAX(INTEGER) to help me understand things!

There are problems with this proposal:

(a) It means that the value of ORD(1) depends on its context. It may be equal to 1 or 32769 depending on whether an INTEGER or CARDINAL value is expected.

(b) If c is a CARDINAL variable having the value 1 then VAL(INTEGER,c) would have the value -32767. Thus, to convert a numerical value from CARDINAL to INTEGER, one would need to use something like VAL(INTEGER,c+32768).

(c) It assumes that the number of values of type INTEGER is not more than the number of values of type CARDINAL.

(d) It will need some amendment to cope with the ordinal numbers of the type LONGINT.

5.2 ORD(-1) has the value -1

Section 6.4.2.2 of the ISO Pascal Standard states that "the ordinal number of a value of integer-type shall be the value itself". Hence, in Pascal, ORD(-1) has the value -1. The major difficulty with using this in Modula-2 is that, in Modula-2, ORD delivers a CARDINAL. Altering ORD to produce an INTEGER would cause problems for examples like ORD(MAX(CARDINAL)).

5.3 ORD(-1) has the value 1

One obvious way of avoiding the negative numbers is to say that "the ordinal number of a value of the type INTEGER shall be its absolute value". This, of course, leads to a problem with VAL(INTEGER,1). Is this equal to -1 or 1?

5.4 ORD(-1) leads to an exception

On behalf of the BSI's Modula-2 Working Group, Don Ward and I have recently been considering the formal definition of Modula-2's standard procedures. We propose that:

(a) ORD delivers a CARDINAL

(b) when n is of some whole-number-type, ORD(n) has the same numerical value as n, no matter what the type of n is provided that the numerical value of n belongs to the type CARDINAL

(c) when n is of some whole-number-type, ORD(n) leads to an exception if the numerical value of n is not a value of the type CARDINAL

This proposal is written in terms of "whole-number-type". Thus, it is applicable not only when n is of type INTEGER but also when n is, say, of the type LONGINT.

6. Feedback

Comments on this paper can be sent to any of the following electronic mail addresses:

JANET: Barry_Cornelius@uk.ac.dur.mts
ARPANET: Barry_Cornelius%mts.dur.ac.uk@cs.ucl.ac.uk
UUCP: ...mcvax!ukc!mts.dur.ac.uk!Barry_Cornelius
BITNET/EARN: Barry_Cornelius%DUR.MTS@AC.UK

Date: Wed, 19 Aug 87 10:34:45 BST
From: Roger Henry <rbh@computer-science.nottingham.ac.uk>
To: info-modula-2@cs.rochester.edu
Subject: Mailing of Proposed BSI Standard Modula-2 I/O Library Def Mod

Over the next few days, five mailings of the Proposed BSI I/O Library Definition Modules will be made. This version has been produced following decisions made by the BSI group IST/5/13 in June 1987 but has not yet been reviewed by that group. The following copyright notice applies in all cases:

```
(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)
```

The five mailings are organised as follows:

```
Numbers of pre-opened channels
1       StdIO           Standard channel number variables

High-level I/O of characters, lines and new-lines
1       Char            I/O of characters
1       Line            I/O of the contents of lines or parts of lines
1       Nl              Skipping and writing of new lines

High-level I/O of basic types with conversion from and to character seq
1       Int             INTEGER I/O with conversion
*       Card            CARDINAL I/O with conversion
*       Real            REAL I/O with conversion
*       Bool            BOOLEAN I/O with conversion

Common conversion modules
1       ConvTypes       Types used in conversions from and to character
1       ConvReports     Handling of input conversion reports

Modules related to character handling and conversions
2       CharIs          predicates for character class testing
2       IntConv         INTEGER/String conversions
*       CardConv        CARDINAL/String conversions
*       RealConv        REAL/String conversions
*       BoolConv        BOOLEAN/String conversions

High-level generic I/O without conversion
%       Gen             Reading, skipping and writing types without con

Identification of channels
3       Channel         Channel numbers and allocation of channel numbe

Non data transfer generic operations on devices linked to channels
3       Device          General properties of/operations on linked devi

Opening devices/device instances and special operations on devices
3       Term            Basic terminal device driver
3       SFile           Sequential files device driver
3       RFile           Random access files device driver

File Directory operations
%       DirOps          Directory operations on closed files (Delete ..

Device independent I/O of storage units and characters
4       In              non-buffered calls of device read routines
4       BuffIn          Buffered input allowing lookahead
4       Out             calls of device write routines

Error handling and termination
5       IOTermination   Actions on program termination
5       IOErrors        device error handling manager

The low-level link between channels and devices
5       Drivers         interface for device drivers and indirect calls
```

```
Notes:
1       first mailing
2       second mailing
3       third mailing
4       fourth mailing
5       fifth mailing
*       Not in current distribution
        (Modelled on corresponding INTEGER modules)
%       Not yet specified
```

For those of you who have not seen earlier versions of the library, you should be aware that many of the module and procedure naming conven assume the use of qualified identifiers.

Trial Implementation Modules are available to implementors under licenc Please contact me if you are interested.

Roger Henry.

Address:
Computer Science Department,
University of Nottingham, Nottingham, NG7 2RD, England
Telephone:
Nottingham (0602 or +44 602) 506101 extension 2855
Electronic Mail Addresses:
JANET: rbh@uk.ac.nott.cs
ARPANET: rbh%cs.nott.ac.uk@UCL-CS.ARPA
UUCP: rbh%cs.nott.ac.uk@ukc.uucp
BITNET/EARN: rbh%NOTT.CS@AC.UK

Date: Wed, 19 Aug 87 10:37:01 BST
To: info-modula-2@cs.rochester.edu
Subject: Number One of Five Mailings of BSI I/O Def Mods

::::::::::::::
StdIO.def
::::::::::::::

```
DEFINITION MODULE StdIO;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(* Standard channel number variables *)

IMPORT
  Channel;

(*
EXPORT QUALIFIED
  in, out, error;
*)

VAR
  in: Channel.Numbers;          (* standard input *)
  out: Channel.Numbers;         (* standard output *)
  error: Channel.Numbers;       (* standard error *)

(*
These variables shall be initialised by the module to the numbers
of preopened channels.
The numbers shall be selected by Channel.Allocate.
*)

(* notes:
Channel "in" may also be open for output, in which case it may be used
to write prompts for input data.
```

```
Other modules may substitute different values for the standard channel
numbers to achieve redirection of standard input and output.
*)

END StdIO.

::::::::::::::
Char.def
::::::::::::::

DEFINITION MODULE Char;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
I/O of characters
*)

IMPORT
  Channel;
FROM ConvTypes IMPORT
  Justifications;

(*
EXPORT QUALIFIED
  Read, Value, Skip,
  CanRead, CanSkip
  Write, WriteF;
*)

(* Read and consume a character: *)

PROCEDURE Read(cn: Channel.Numbers; VAR char: CHAR);
(* pre  : there is at least one character in the input *)
(* post : the next character is stored in "char", *)
(*        and has been skipped over in the input. *)

(* The next character may be obtained without consuming the input: *)

PROCEDURE Value(cn: Channel.Numbers): CHAR;
(* pre  : there is at least one character in the input *)
(* post : the next character is stored in "char". *)

(* If the character is not going to be read it may be skipped: *)

PROCEDURE Skip(cn: Channel.Numbers);
(* pre  : there is at least one character in the input *)
(* post : the next character has been skipped *)

(*
If the preconditions of Read, Value, or Skip are not satisfied,
a conversion error shall be reported via ConvReports:
  noData: end of input is found before any character.

Initially a handler shall be installed which causes a message to be pri
and the program to be terminated.
A handler may be installed which allows continuation and subsequent
testing for errors.
Any returned value shall be 0C.

Alternatively, predicates can be called to test the precondition in adv
*)

PROCEDURE CanRead(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff there is at least one character in the input
(*        The reason for a FALSE result is given by ConvReports.LastRep
(* note : this can only be "noData" *)

PROCEDURE CanSkip(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff there is at least one character in the input
(*        The reason for a FALSE result is given by ConvReports.LastRep
(* note : this can only be "noData" *)

(*
*)

PROCEDURE Write(cn: Channel.Numbers; char: CHAR);
(* pre  : the channel is open for output operations *)
(* post : the given character value is written to the Linked device *)

PROCEDURE WriteF(
      cn: Channel.Numbers;
      char: CHAR;
      width: CARDINAL;
      where: Justifications
  );
(* pre  : the channel is open for output operations *)
(* post : the given value is written in a field of the given *)
(*        minimum width left, centre, or right justified *)

END Char.

::::::::::::::
Line.def
::::::::::::::

DEFINITION MODULE Line;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
I/O of the contents of lines or parts of lines
*)

IMPORT
  Channel;

(*
EXPORT QUALIFIED
  Read, GetValue, Skip,
  CanRead, CanSkip
  Write;
*)

(* Read and consume characters before a new line: *)

PROCEDURE Read(cn: Channel.Numbers; VAR chars: ARRAY OF CHAR);
(* pre  : there is at least one character before a new line (Nl) *)
(* do   : read characters until the given array fills or Nl found *)
(* post : the characters are stored in the array, *)
(*        with a 0C terminator if necesary, *)
(*        and have been skipped over in the input. *)
(* note : the Nl character is not stored or consumed *)

(* The characters may be obtained without consuming the input: *)

PROCEDURE GetValue(cn: Channel.Numbers; VAR chars: ARRAY OF CHAR);
(* pre  : there is at least one character before a new line (Nl) *)
(* do   : read characters until the given array fills or Nl found *)
(* post : the characters are stored in the array, *)
(*        with a 0C terminator if necesary. *)
(* note : the Nl character is not stored *)

(* If the rest of the line is not going to be read it may be skipped: *
```

```
PROCEDURE Skip(cn: Channel.Numbers);
(* pre  : there is at least one character before a new line (Nl) *)
(* post : all characters upto but excluding the next Nl have been skipp

(*
If the preconditions of Read, GetValue, or Skip are not satisfied,
a conversion error shall be reported via ConvReports:
  foundNl: Nl found before any other characters,
  noData: end of input is found before any characters.

Initially a handler shall be installed which causes a message to be pri
and the program to be terminated.
A handler may be installed which allows continuation and subsequent
testing for errors.

Alternatively, predicates can be called to test the precondition in adv
*)

PROCEDURE CanRead(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff there is at least one character *)
(*        before a new line (Nl) *)
(*        The reason for a FALSE result is given by ConvReports.LastRep

PROCEDURE CanSkip(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff there is at least one character *)
(*        before a new line (Nl) *)
(*        The reason for a FALSE result is given by ConvReports.LastRep

PROCEDURE Write(cn: Channel.Numbers; chars: ARRAY OF CHAR);
(* pre  : the channel is open for output operations *)
(* post : the given string value is written to the Linked device *)

END Line.

::::::::::::::
Nl.def
::::::::::::::

DEFINITION MODULE Nl;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
 *)

(*
Skipping and writing of new lines
*)

IMPORT
  Channel;

(*
EXPORT QUALIFIED
  Skip, CanSkip, Write;
*)

PROCEDURE Skip(cn: Channel.Numbers);
(* pre  : the next significant character is new line (Nl) *)
(* do   : skip characters until the Nl is passed *)

(*
If the precondition of Skip is not satisfied,
a conversion error shall be reported via ConvReports:
  badFormat: another significant character found before Nl,
  noData: end of input is found before Nl.

Initially a handler shall be installed which causes a message to be pri
and the program to be terminated.
A handler may be installed which allows continuation and subsequent
testing for errors.
Significant characters shall not have been consumed.

Alternatively, a predicate can be called to test the precondition in ad
*)

PROCEDURE CanSkip(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff Nl is the next significant character *)
(*        The reason for a FALSE result is given by ConvReports.LastRep

PROCEDURE Write(cn: Channel.Numbers);
(* pre  : the channel is open for output operations *)
(* do   : write the Nl character to the Linked device *)

END Nl.

::::::::::::::
Int.def
::::::::::::::

DEFINITION MODULE Int;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
 *)

(*
INTEGER IO with conversion from and to character sequences
*)

IMPORT
  Channel;
FROM ConvTypes IMPORT
  Justifications;

(*
EXPORT QUALIFIED
  Read, Value, Skip,
  CanRead, CanSkip
  Write, WriteF;
*)

(* Input:
leading white-space (Space and Tab) ignored,
optional +/- sign,
followed immediately by a sequence of decimal digits,
terminated by the first non-digit.
The terminating character is not consumed.
*)

(* Read and consume a legible integer value *)

PROCEDURE Read(cn: Channel.Numbers; VAR int: INTEGER);
(* pre  : there is a well-formed integer next in the input data *)
(*        which can be represented as an INTEGER value *)
(* post : value of "int" = what was the next data value and this has be
(*        skipped over in the input *)

(* The value of the integer can be obtained without consuming the input

PROCEDURE Value(cn: Channel.Numbers): INTEGER;
(* pre  : there is a well-formed integer next in the input data *)
(*        which can be represented as an INTEGER value *)
(* post : returns the value of the next item of data *)

(* If the integer is not going to be read it may be skipped: *)
```

```
PROCEDURE Skip(cn: Channel.Numbers);
(* pre  : there is a well-formed integer next in the input data *)
(* post : what was the next data value has been skipped over *)

(*
If the preconditions of Read, Value, or Skip are not satisfied,
a conversion error shall be reported via ConvReports:
  badValue: the value is too large to be represented,
  badFormat: next characters within line do not give a well-formed inte
  foundNl: unexpected new line character found,
  noData: end of input is found before any significant characters.

Initially a handler shall be installed which causes a message to be pri
and the program to be terminated.
A handler may be installed which allows continuation and subsequent
testing for errors.
Significant characters shall not have been consumed,
and any returned value shall be MAX(INTEGER).

Alternatively, predicates can be called to test the precondition in adv
*)

PROCEDURE CanRead(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff there is a well-formed integer next in the i
(*        data which can be represented as an INTEGER value. *)
(*        The reason for a FALSE result is given by ConvReports.LastRep

(*
Skipping does not strictly require the value to be representable:
*)

PROCEDURE CanSkip(cn: Channel.Numbers): BOOLEAN;
(* pre  : the channel is open for input operations *)
(* post : returns TRUE iff there is a well-formed integer next in the i
(*        The reason for a FALSE result is given by ConvReports.LastRep

(* Output:
no specified field width:
  leading sign for negative integers, space for non-negative.
with specified field width:
  leading sign for negative integers, space for non-negative,
  left, centre, or right justified within the given minimum field width
  In the special case of a specified field width of 0, the leading spac
  for non-negative values shall be suppressed.
*)

PROCEDURE Write(cn: Channel.Numbers; int: INTEGER);
(* pre  : the channel is open for output operations *)
(* post : the given integer value is written with no field *)
(* note : successive values written in this format can be distinguished
(*        on input *)

PROCEDURE WriteF(
      cn: Channel.Numbers;
      int: INTEGER;
      width: CARDINAL;
      where: Justifications
  );
(* pre  : the channel is open for output operations *)
(* post : the given value is written in a field of the given *)
(*        minimum width left, centre, or right justified *)

END Int.

::::::::::::::
ConvTypes.def
::::::::::::::

DEFINITION MODULE ConvTypes;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact. for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transfer
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(* Types used in conversions from and to character sequences *)

(*
EXPORT QUALIFIED
  ConvResults, BadConvResults, Justifications;
*)

TYPE
  ConvResults = (
    goodValue,          (* good format and value *)
    badValue,           (* good format but value cannot be represented
    badFormat,          (* bad format of data within line *)
    foundNl,            (* unexpected new line character found *)
    noData              (* no significant characters to convert *)
    );
  BadConvResults = [badValue .. noData];
  Justifications = (left, centre, right);

END ConvTypes.

::::::::::::::
ConvReports.def
::::::::::::::

DEFINITION MODULE ConvReports;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Handling of input conversion reports
*)

IMPORT
  Channel;
FROM ConvTypes IMPORT
  ConvResults, BadConvResults;

(*
EXPORT QUALIFIED
  NoteGood, NoteBad, NoteError,
  Handlers,
  Abort, Continue, GetHandler, SetHandler,
  LastReport;
*)

(* An <Object>IO input procedure - Read, Value, or Skip - reports when
(* it has a good result on a channel: *)

PROCEDURE NoteGood(cn: Channel.Numbers);
(* post: the result of the last input procedure is noted as good *)

(* An <Object>IO input predicate - CanRead, or CanSkip - reports the *)
(* reason when it returns FALSE: *)

PROCEDURE NoteBad(cn: Channel.Numbers; badRes: BadConvResults);
(* post: the result of the last input test is noted as bad *)

(* The <Object>IO input procedures report when an incorrect format or
(* out of range value condition arises on input. *)
(* When reported, a per-channel handler routine is called: *)

PROCEDURE NoteError(
      cn: Channel.Numbers;
```

```
      badRes: BadConvResults;
      msg: ARRAY OF CHAR);
(* pre : badRes conveys the problem. *)
(*       msg gives further information such as calling procedure name *
(* post: the condition has been noted and a per-channel handler called

TYPE
  Handlers =
    PROCEDURE(Channel.Numbers, BadConvResults, VAR ARRAY OF CHAR);

(* The reported message shall be passed unchanged to the handler. *)
(* Initially a handler is installed which composes a message *)
(* and terminates the program. *)

(* The standard handlers: *)

PROCEDURE Abort(
      cn: Channel.Numbers;
      res: BadConvResults;
  VAR msg: ARRAY OF CHAR
  );
(* default handler *)
(* print message from res and msg and terminate the program *)

PROCEDURE Continue(
      cn: Channel.Numbers;
      res: BadConvResults;
  VAR msg: ARRAY OF CHAR
  );
(* alternative standard handler *)
(* do nothing *)

(*
Get and set the input conversion error handler for a channel
*)

PROCEDURE GetHandler(cn: Channel.Numbers; VAR h: Handlers);
(* post: the current handler is assigned to h *)

PROCEDURE SetHandler(cn: Channel.Numbers; h: Handlers);
(* post: the current handler is set to h *)

(*
Allow the user to test if a call for legible input was successful,
in cases where the handler has continued.
Determine the reason for a FALSE result from an input predicate
*)

PROCEDURE LastReport(cn: Channel.Numbers): ConvResults;
(* pre  : there has been a call of an <Object>IO input routine *)
(*        or a call of a predicate which returned FALSE *)
(* post : returns the report made on the last such call on the channel

END ConvReports.

Date:     Wed, 19 Aug 87 17:31:00 BST
To: info-modula-2@cs.rochester.edu
Subject:  Number 2 of five Mailings of Proposed BSI Standard I/O Librar

::::::::::::::
CharIs.def
::::::::::::::

DEFINITION MODULE CharIs;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
 *)

(*
Predicates for character class testing
*)

(*
EXPORT QUALIFIED
  Nl, Digit, Space, Sign, Upper, Lower;
*)

PROCEDURE Nl(ch: CHAR): BOOLEAN;
(* post : returns TRUE iff "ch" is the implementations new line charact

PROCEDURE Digit(ch: CHAR): BOOLEAN;
(* post : returns TRUE iff "ch" is a decimal digit *)

PROCEDURE Space(ch: CHAR): BOOLEAN;
(* post : returns TRUE iff "ch" is a whitespace character (space or tab

PROCEDURE Sign(ch: CHAR): BOOLEAN;
(* post : returns TRUE iff "ch" is + or - sign *)

PROCEDURE Upper(ch: CHAR): BOOLEAN;
(* post : returns TRUE iff "ch" is an upper case letter *)

PROCEDURE Lower(ch: CHAR): BOOLEAN;
(* post : returns TRUE iff "ch" is a lower case letter *)

END CharIs.

::::::::::::::
IntConv.def
::::::::::::::

DEFINITION MODULE IntConv;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
 *)

(*
Integer/String conversions.
*)

FROM ConvTypes IMPORT
  ConvResults, Justifications;

(*
EXPORT QUALIFIED
  FromStr, ToStr, ToField;
*)

(* well-formed character representation:
any number of leading white-space characters but not NlCh,
optional +/- sign,
followed immediately by a sequence of decimal digits,
terminated by the first non-digit or end of array.
*)

PROCEDURE FromStr(
  VAR str: ARRAY OF CHAR;
  VAR index: CARDINAL;
  VAR int: INTEGER;
  VAR res: ConvResults
  );
(* pre  : "index" gives position from which to start conversion *)
(* post : "res" gives result of conversion *)
(*        iff "res" is goodValue *)
(*          converted value stored in "int", and *)
(*          "index" updated to position after last character of integer

(* no specified field width:
```

```
  leading sign for negative integers, space for        non-negative,
  0C terminator stored if room in array.
*)

PROCEDURE ToStr(
      int: INTEGER;
  VAR str: ARRAY OF CHAR;
  VAR index: CARDINAL
  );
(* pre  : "index" is position relative to start of "str" at which to *)
(*        store character representation of "int" *)
(* post : "index" = old "index" + number of characters in representatio
(*        Characters stored at corresponding positions in "str" if with
(*        the array bounds, including 0C at str[index]. *)

(* with specified field width:
  leading sign for negative integers, space for non-negative,
  left, centre, or right justified within the given minimum field width
  In the special case of a specified field width of 0, the leading spac
  for non-negative values shall be suppressed.
*)

PROCEDURE ToField(
      int: INTEGER;
  VAR str: ARRAY OF CHAR;
  VAR index: CARDINAL;
      width: CARDINAL;
      where: Justifications
  );
(* pre  : "index" is position relative to start of "str" at which to *)
(*        store character representation of "int" *)
(* post : "index" = old "index" + number of characters in representatio
(*        Characters stored at corresponding positions in "str" if with
(*        the array bounds. *)

END IntConv.
```

Date: Thu, 20 Aug 87 10:58:27 BST
To: info-modula-2@cs.rochester.edu
Subject: Number 3 of five mailings of the proposed BSI I/O Library Def

```
::::::::::::::
Channel.def
::::::::::::::

DEFINITION MODULE Channel;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Identification of channels by channel numbers.
Allocation of channel numbers.
*)

(*
EXPORT QUALIFIED
  noNumber, minNumber, maxReady, maxNumber, Numbers, AllNumbers,
  Allocate, IsAllocated, Deallocate;
*)

CONST
  noNumber = 0;                    (* value if none free *)
  minNumber = 1;                   (* lowest channel number *)
  maxReady = 10;                   (* highest pre-allocated *)
  maxNumber = 64;                  (* highest channel number *)

(* maxReady and maxNumber are implementation-defined values *)

  TYPE
    Numbers = [minNumber .. maxNumber];
    AllNumbers = [noNumber .. maxNumber];

(* Channel numbers minNumber to maxReady are always allocated *)
(* Channel numbers maxReady+1 to maxNumber are initially unallocated *)

(* note:
the pre-allocated numbers may be used in simple programs and other
contexts where the user can ensure that there will be no conflicts.
*)

PROCEDURE Allocate(VAR cn: AllNumbers);
(* post: cn is set to noNumber *)
(*       or to a previously unallocated channel number *)
(*       which is now allocated *)

(* note:
the dynamic allocation scheme is provided for use in independent module
or other contexts where the numbers in use are not known
*)

PROCEDURE IsAllocated(cn: Numbers): BOOLEAN;
(* post: TRUE iff the given channel number is allocated *)

PROCEDURE Deallocate(cn: Numbers);
(* post : for cn > maxReady, the channel number is now not Allocated *)
(*        pre-allocated channels remain allocated *)

END Channel.
::::::::::::::
Device.def
::::::::::::::

DEFINITION MODULE Device;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
General properties of devices linked to a channel.
Operations on the devices.
*)

IMPORT
  Channel;

(*
EXPORT QUALIFIED
  Modes, Mappings, NlCh,
  IsOpen, IsReadable, IsWriteable, IsText, IsInteractive,
  Flush, Reset, Close;
*)

(* Channels are not initially Linked to a device. *)
(* The Linking is done by procedures exported from device driver module
(* These procedures specify, or allow the choice of, one of the *)
(* following Modes and Mappings *)

TYPE
  Modes = (readOnly, writeOnly, readWrite);

  Mappings = (text, binary);
  (* text mapping device drivers shall apply any necessary *)
  (* mapping between the external representation  of text files *)
  (* and the internal representation. *)
  (* The internal representation shall be a sequence *)
  (* of storage units interpreted as CHAR values with new lines being *
```

```
  (* represented by a single implementation-defined CHAR value.
  (* In the case of binary mapping device drivers, the internal *)
  (* representation of data shall be a sequence of storage units *)
  (* corresponding directly to the external sequence of storage units.

  (* note:
  Not all operating systems distinguish between text and binary files.
  However, for portability of programs between operating systems, text
  mapping must be selected if the external data has to be in a format
  compatible with other text files in the environment.
  An application may employ binary mapping when it
  only needs to write text to be read by itself
  (or by other applications using this library on the same system).
  *)

PROCEDURE NlCh(): CHAR;
(* post : returns the character used to represent new lines internally

(* A channel is Open if it is Linked to a device *)
(* Closed means not Open *)

PROCEDURE IsOpen(cn: Channel.Numbers): BOOLEAN;
(* post : TRUE iff the numbered channel is Linked to a device *)

PROCEDURE IsReadable(cn: Channel.Numbers): BOOLEAN;
(* post : TRUE iff the numbered channel is Linked to a device for input

PROCEDURE IsWriteable(cn: Channel.Numbers): BOOLEAN;
(* post : TRUE iff the numbered channel is Linked to a device for outpu

PROCEDURE IsText(cn: Channel.Numbers): BOOLEAN;
(* pre  : the numbered channel is Open *)
(* post : TRUE iff the numbered channel conforms to text mapping rules

(* The user may enquire if a device is interactive - *)
(* if it is then input is being generated while the program is running
(* rather than being stored as in a disc file *)

PROCEDURE IsInteractive(cn: Channel.Numbers): BOOLEAN;
(* pre  : the numbered channel is Open *)
(* post : returns TRUE iff the device is interactive *)

PROCEDURE Flush(cn: Channel.Numbers);
(* pre  : the numbered channel is Linked to a device Open for output *)
(* post : all data held in output buffers has been written *)

PROCEDURE Reset(cn: Channel.Numbers);
(* pre  : the numbered channel is Open *)
(* post : the drivers and the device or device instance are reset *)

PROCEDURE Close(cn: Channel.Numbers);
(* pre  : the numbered channel is Open *)
(* post : all data held in output buffers has been written *)
(* note : the channel is no longer linked to the device *)

END Device.

::::::::::::::
Term.def
::::::::::::::

DEFINITION MODULE Term;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(* Terminal device driver module *)

IMPORT
  Channel;
*)

(*
EXPORT QUALIFIED
  ByChar, ByLine, IsMine;
*)

(*
Char mode:
        input is available as soon as it has been typed;
        characters read from the channel are not echoed by the driver;
        a read request for n characters will only return 1 character;
        text (line) mapping applies.

Line mode:
        input is buffered and made available on typing the appropriate
        end of line indication or some system-defined character
        (which will not be included in the input);
        characters read from the channel are echoed by the driver;
        a read request for n characters will return m characters
        as soon as they are available where 0 < m <= n;
        a read request will return 0 characters iff the user has indica
        that there is no more input and all previously typed characters
        have been read - more input may be available after a call of
        Reset on the associated channel;
        text (line) mapping applies.

Typed characters are distributed between channels according to the
sequence of read requests.
Channels Linked to Term are Open for reading and writing and the device
is interactive.
*)

PROCEDURE ByChar(cn: Channel.Numbers);
(* pre : cn is the number of a Closed channel *)
(* post: the channel is Linked to the terminal in Char mode *)

PROCEDURE ByLine(cn: Channel.Numbers);
(* pre : cn is the number of a Closed channel *)
(* post: the channel is Linked to the terminal in Line mode *)

PROCEDURE IsMine(cn: Channel.Numbers): BOOLEAN;
(* returns TRUE iff channel is linked to Term *)

END Term.

::::::::::::::
SFile.def
::::::::::::::

DEFINITION MODULE SFile;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Sequential Files.
There is a common read/write mark associated with every channel.
The mark is at the start of the file after opening and after a Reset.
It is moved forward by the number of storage units read or written.
*)

IMPORT
  Channel;
FROM Device
  IMPORT text, binary, readOnly, readWrite;

(*
EXPORT QUALIFIED
```

```
  Mappings, Modes, New,        Old, Fresh, IsMine;
*)

TYPE
  Mappings = [text .. binary];
  Modes = [readOnly .. readWrite];

(* The Open routines may generate device errors *)

PROCEDURE New(
      cn: Channel.Numbers;
      name: ARRAY OF CHAR;
      map: Mappings;
      mode: Modes
  );
(* pre:  cn is the number of a Closed channel *)
(*       name is the required name of a new file *)
(*       map is text to guarantee line mapping *)
(*       mode gives the required transfer operations *)
(* post: the channel is Linked to SFile *)

PROCEDURE Old(
      cn: Channel.Numbers;
      name: ARRAY OF CHAR;
      map: Mappings;
      mode: Modes
  );
(* pre:  cn is the number of a Closed channel *)
(*       name is the required name of an existing file *)
(*       map is text to guarantee line mapping *)
(*       mode gives the required transfer operations *)
(* post: the channel is Linked to SFile *)

PROCEDURE Fresh(
      cn: Channel.Numbers;
      name: ARRAY OF CHAR;
      map: Mappings;
      mode: Modes
  );
(* pre:  cn is the number of a Closed channel *)
(*       name is the required name of the file *)
(*       map is text to guarantee line mapping *)
(*       mode gives the required transfer operations *)
(* post: the channel is Linked to SFile *)
(*       if the file already existed, it is truncated to zero length *)

PROCEDURE IsMine(cn: Channel.Numbers): BOOLEAN;
(* post: TRUE iff the numbered channel is Linked to SFile *)

END SFile.

::::::::::::::
RFile.def
::::::::::::::

DEFINITION MODULE RFile;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Random access files.
(On some systems, random access may not be supported on text files.)
There is a common read/write mark associated with every channel.
The mark is at the start of the file after opening and after a Reset.
It is moved forward by the number of storage units read or written and
may be saved and restored using GetMark and SetMark.
*)

      IMPORT
  Channel;
FROM Device
  IMPORT text, binary, readOnly, readWrite;

(*
EXPORT QUALIFIED
  Mappings, Modes, New, Old, Fresh, IsMine,
  FileMark, GetMark, SetMark, SetAtStart, SetAtEnd, MoveMark;
*)

TYPE
  Mappings = [text .. binary];
  Modes = [readOnly .. readWrite];

(* The Open routines may generate device errors *)

PROCEDURE New(
      cn: Channel.Numbers;
      name: ARRAY OF CHAR;
      map: Mappings;
      mode: Modes
  );
(* pre:  cn is the number of a Closed channel *)
(*       name is the required name of a new file *)
(*       map is text to guarantee line mapping *)
(*       mode gives the required transfer operations *)
(* post: the channel is Linked to RFile *)

PROCEDURE Old(
      cn: Channel.Numbers;
      name: ARRAY OF CHAR;
      map: Mappings;
      mode: Modes
  );
(* pre:  cn is the number of a Closed channel *)
(*       name is the required name of an existing file *)
(*       map is text to guarantee line mapping *)
(*       mode gives the required transfer operations *)
(* post: the channel is Linked to RFile *)

PROCEDURE Fresh(
      cn: Channel.Numbers;
      name: ARRAY OF CHAR;
      map: Mappings;
      mode: Modes
  );
(* pre:  cn is the number of a Closed channel *)
(*       name is the required name of the file *)
(*       map is text to guarantee line mapping *)
(*       mode gives the required transfer operations *)
(* post: the channel is Linked to RFile *)
(*       if the file already existed, it is truncated to zero length *)

PROCEDURE IsMine(cn: Channel.Numbers): BOOLEAN;
(* post: TRUE iff the numbered channel is Linked to RFile *)

(*
read/write positions are given in storage units relative to the start
of the file.
*)

TYPE
  FileMark =
    RECORD
      high: CARDINAL;
      low: CARDINAL
    END;

PROCEDURE GetMark(cn: Channel.Numbers; VAR mark: FileMark);
(* pre  : the numbered channel is Linked to RFile and is Open *)
(* post : mark = read/write position relative to the start of the file

PROCEDURE SetMark(cn: Channel.Numbers; mark: FileMark);
(* pre  : the numbered channel is Linked to RFile and is Open, *)
(*        mark <= number of storage units in the file *)
```

```
(* post        : the read/write mark is set relative to the start of the file

PROCEDURE SetAtStart(cn: Channel.Numbers);
(* pre  : the numbered channel is Linked to RFile and is Open, *)
(* post : the read/write mark is set to the start of the file *)

PROCEDURE SetAtEnd(cn: Channel.Numbers);
(* pre  : the numbered channel is Linked to RFile and is Open, *)
(* post : the read/write mark is set to the end of the file *)

PROCEDURE MoveMark(cn: Channel.Numbers; by: INTEGER);
(* pre  : the numbered channel is Linked to RFile and is Open, *)
(*        the move will not attempt to move the file mark *)
(*        beyond the limits of the file *)
(* post : the read/write mark is set relative to the previous mark *)

END RFile.
```

Date: Thu, 20 Aug 87 17:07:03 BST
To: info-modula-2@cs.rochester.edu
Subject: Number 4 of Five Mailings of the Proposed BSI I/O Library Def

```
::::::::::::::
In.def
::::::::::::::

DEFINITION MODULE In;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(* Device independent input of storage units and characters *)
(* Note:
This is a direct interface to the device drivers and should not
normally be used on channels over which higher level buffered input
is taking place.
*)

FROM SYSTEM IMPORT
  ADDRESS;
IMPORT
  Channel;

(*
EXPORT QUALIFIED
  Read, ReadCh, ReadChars;
*)

PROCEDURE Read(
      cn: Channel.Numbers;
      maxStunts: CARDINAL;        (* maximum number of storage units to r
      where: ADDRESS;             (* where to read them to *)
  VAR nStunts: CARDINAL);         (* number read *)
(* pre : the numbered channel is open for reading *)
(* do  : call Linked device read routine once *)
(* post: nStunts is the number of storage units supplied by Linked devi
(*       nStunts = 0 implies end of data *)

PROCEDURE ReadCh(cn: Channel.Numbers; VAR ch: CHAR; VAR done: BOOLEAN);
(* pre : the numbered channel is open for reading *)
(* do  : read and interpret storage units from device for one character
(* post: iff done = TRUE, a character has been read and stored in ch *)
(* note: done is FALSE if there is not enough data from the device *)

PROCEDURE ReadChars(
      cn: Channel.Numbers;
  VAR chars: ARRAY OF CHAR;       (* array to read chars to *)
      start: CARDINAL;            (* relative index to start storing *)
  VAR nChars: CARDINAL            (* number of characters read *)
);
(* pre : the numbered channel is open for reading *)
(* do  : call Linked driver to read characters from start to end of arr
(* post: nChars is the number of characters supplied by Linked device *
(*       nChars = 0 implies end of data *)

END In.
::::::::::::::
BuffIn.def
::::::::::::::
DEFINITION MODULE BuffIn;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

IMPORT
  Channel;
FROM SYSTEM IMPORT
  ADDRESS;

(*
EXPORT QUALIFIED
  Have, Look, Forward,
  HaveCh, ThisCh, ForwardCh,
  Mark, Rewind,
  SetBuffer, FreeBuffer;
*)

(*
Input is taken through a buffer.
The buffer is allocated on the first operation on which it is needed
(in which case a default buffer size is used) or it may be pre-allocate

Look and ThisCh copy data from the current read position.
The read position is only moved forward by calls of Forward and Forward
It is returned to the logical start of the buffer by Rewind.
Mark sets the logical start of the buffer to the current read position.

The buffer shall be set empty if the driver indicates that the read pos
has moved since the last read (because of an intervening operation such
a seek on a file).
*)

PROCEDURE Have(cn: Channel.Numbers; nStunts: CARDINAL): BOOLEAN;
(* pre : the numbered channel is open for reading *)
(* do  : make repeated calls of the Linked driver routines as necessary
(*       to make nStunts storage units available in the buffer or *)
(*       until no further data is available from the device *)
(* post: returns TRUE iff there are at least nStunts of data *)
(*       available from the current read position *)

PROCEDURE Look(cn: Channel.Numbers; nStunts: CARDINAL; where: ADDRESS);
(* pre  : at least nStunts of data are available from the *)
(*        current read position in the buffer or in the subsequent data
(* post : nStunts storage units copied from the read position *)
(*        to the given address *)

PROCEDURE Forward(cn: Channel.Numbers; nStunts: CARDINAL);
(* pre  : at least nStunts of data are available from the *)
(*        current read position in the buffer or in the subsequent data
(* post : the read position is moved forward by nStunts storage units *

(*
Character routines also provided for convenience
*)

PROCEDURE HaveCh(cn: Channel.Numbers): BOOLEAN;
(* post : returns TRUE iff there is at least one character available *)
(*        from the current read position *)
```

```
PROCEDURE ThisCh(cn: Channel.Numbers): CHAR;
(* pre  : at least one  character is available from the current *)
(*        read position *)
(* post : returns the character at the current read position *)

PROCEDURE ForwardCh(cn: Channel.Numbers);
(* pre  : at least one  character is available from the current *)
(*        read position *)
(* post : the read position is moved forward by one character *)

PROCEDURE Mark(cn: Channel.Numbers);
(* pre  : a buffer is allocated for the channel *)
(* post : the start of the buffer is set to the current read position *

PROCEDURE Rewind(cn: Channel.Numbers);
(* pre  : a buffer is allocated for the channel *)
(* post : the current read position is set to the buffer start *)

PROCEDURE SetBuffer(cn: Channel.Numbers; bufSize: CARDINAL);
(* pre  : a buffer is not allocated for the channel *)
(* post : a buffer of the given size is allocated to the channel *)

PROCEDURE FreeBuffer(cn: Channel.Numbers);
(* post : any buffer previously allocated to the channel is now freed *
(* note : data which has not been copied from the buffer will be lost *

END BuffIn.

::::::::::::::
Out.def
::::::::::::::

DEFINITION MODULE Out;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(* device independent output of storage units and characters *)

IMPORT
  Channel;
FROM SYSTEM IMPORT
  ADDRESS;

(*
EXPORT QUALIFIED
  Write, WriteCh, WriteChars;
*)

PROCEDURE Write(
      cn: Channel.Numbers;
      nStunts: CARDINAL;          (* number of storage units to write *)
      where: ADDRESS              (* from where *)
  );
(* pre  : the numbered channel is open for writing *)
(* post : nStunts storage units have been copied by the device driver *
(*        for transmission to the destination *)

PROCEDURE WriteCh(cn: Channel.Numbers; ch: CHAR);
(* pre : the numbered channel is open for writing *)
(* post: the given character value has been copied by the device driver
(*       for transmission to the destination *)

PROCEDURE WriteChars(
      cn: Channel.Numbers;
  VAR chars: ARRAY OF CHAR;
      start, nChars: CARDINAL
  );
(* pre : the numbered channel is open for writing *)
(*       there are nChars initialised elements from start in the array
(* post: nChars character values from start have been copied by *)
(*       the device driver for transmission to the destination *)

END Out.
```

Date: Thu, 20 Aug 87 17:57:33 BST
To: info-modula-2@cs.rochester.edu
Subject: Number 5 of 5 mailings of the proposed BSI I/O Library def mo

```
::::::::::::::
IOTermination.def
::::::::::::::

DEFINITION MODULE IOTermination;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Actions on program termination
*)

IMPORT
  Channel;

(*
EXPORT QUALIFIED
  Handler, Establish, Terminate;
*)

TYPE
  Private;
  Handler =
    RECORD
      code: PROC;
      data: Private;
    END;

(* There is a list of procedures to be called on program termination *)
(* initially the list is empty *)

PROCEDURE Establish(VAR h: Handler);
(* post: the given procedure is on the front of the list of those which
(*       will be called on termination *)

PROCEDURE Terminate(cn: Channel.Numbers; source, message: ARRAY OF CHAR
(* displays channel number, source module.procedure and message *)
(* calls established procedures in the order of the list *)
(* halts the program *)

(* notes:
If module A imports module B, and both call Establish
from their initialisation part, B will be initialised before A
and terminated after A.  This is normally safe unless B calls
procedures of A through procedure variables.

If the Modula-2 system provides a general termination facility,
this module should establish its own general termination code
which will implicitly invoke procedures on its list of handlers
*)

END IOTermination.

::::::::::::::
IOErrors.def
::::::::::::::
```

```
DEFINITION MODULE IOErrors;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Device Error Handling Manager
*)

IMPORT
  Channel;

(*
EXPORT QUALIFIED
  ErrorHandlers, Abort, Continue,
  GetErrorHandler, SetErrorHandler,
  DeviceOK, DeviceError,
  ReportOK, ReportError;
*)

(* Device driver routines are required to report when *)
(* their error state changes. *)
(* The state of all channels is initiall OK *)
(* On the report of an error, *)
(* a per-channel handler routine is called. *)
(* In itially a handler is installed which causes program termination.
(* Post conditions given in the specifications of all library procedure
(* which call device driver routines assume that the default handler is
(* installed or that there has been no device error *)

TYPE
  ErrorHandlers =
    PROCEDURE(
          Channel.Numbers,        (* number of Linked channel *)
          INTEGER,                (* device dependent error number *)
      VAR ARRAY OF CHAR,          (* source module.procedure name *)
      VAR ARRAY OF CHAR           (* descriptive error message *)
    );

(*
The standard error handlers
*)

PROCEDURE Abort(
      cn: Channel.Numbers;
      errNum: INTEGER;
  VAR source: ARRAY OF CHAR;
  VAR msg: ARRAY OF CHAR
);
(* default error handler *)
(* shows messages and terminates the program *)

PROCEDURE Continue(
      cn: Channel.Numbers;
      errNum: INTEGER;
  VAR source: ARRAY OF CHAR;
  VAR msg: ARRAY OF CHAR
);
(* alternative standard error handler *)
(* does nothing *)

(*
The user can get and set the error handler for a channel:
*)

PROCEDURE GetErrorHandler(cn: Channel.Numbers; VAR eh: ErrorHandlers);
(* post: the current handler for the channel is assigned to eh *)

PROCEDURE SetErrorHandler(cn: Channel.Numbers; eh: ErrorHandlers);
(* post: the current handler for the channel is set to eh *)

(*
The user can test if an error has been reported.
Used in cases where the handler has continued.
*)

PROCEDURE DeviceOK(cn: Channel.Numbers): BOOLEAN;
(* post : TRUE initially and if the associated device is ok *)

(*
The error number is remembered by the manager
*)

PROCEDURE DeviceError(cn: Channel.Numbers): INTEGER;
(* pre  : the numbered channel is not ok *)
(* post : returns the latest reported device-specific error number *)

(* note:
If the user wishes to have the error message remembered,
so that it may be shown after continuation and return from the calling
routine, she or he may install their own handler which does this -
on a global or per channel basis as appropriate.
*)

(*
Drivers must report changes in the error status:
*)

PROCEDURE ReportOK(cn: Channel.Numbers);
(* post : the error state of the Linked device is noted as OK *)

PROCEDURE ReportError(
      cn: Channel.Numbers;
      errNum: INTEGER;
  VAR source: ARRAY OF CHAR;
  VAR msg: ARRAY OF CHAR
);
(* post : the error condition has been reported by calling the handler
(*        for the channel. *)
(*         The error state and number have been noted *)

END IOErrors.

::::::::::::::
Drivers.def
::::::::::::::

DEFINITION MODULE Drivers;

(* Proposed BSI Standard Modula-2 I/O Library
 * Copyright Roger Henry, University of Nottingham
 * Version WG/2.0, August 17th 1987
 * Permission is given to copy this Definition Module, with the
 * copyright notice intact, for the purposes of evaluation and test.
 * At the stage of a formal draft standard, Copyright will be transferr
 * to BSI (and through BSI to other recognised standards bodies).
 * Status: For review by BSI/IST/5/13
*)

(*
Interface for device drivers and indirect callers of driver routines
*)

IMPORT
  Channel;
FROM SYSTEM IMPORT
  ADDRESS;

(*
EXPORT QUALIFIED
  Conditions, TestProc,
  ReadProc, WriteProc, ResetProc, FlushProc,
  CloseProc, FreeProc,
  DriverProcs, DriverRefs,
  MakeLink,
```

```
  ToTest, ToRead, ToWrite, ToReset, ToFlush,
  ToClose, ToFree,
  IsLinked, BreakLink;
*)

(* a table of driver procedures is linked to a channel when *)
(* the channel is open to a device or device instance *)
(* The precise semantics of the operations in particular cases *)
(* shall be documented with the corresponding device modules. *)

TYPE
  Conditions =
    (canRead,                       (* input operations can be tried *)
     canWrite,                      (* output operations can be tried *)
     textMap,                       (* conforms to text mapping rules *)
     isInteractive,                 (* a terminal for example. *)
     readPosMoved                   (* (by reset or seek) since last Read *
    );

  TestProc =                        (* test if condition applies *)
    PROCEDURE(
          Channel.Numbers,
          Conditions
    ): BOOLEAN;

  ReadProc =                        (* if canRead, read data else terminate
    PROCEDURE(
          Channel.Numbers,
          CARDINAL,                 (* max storage units to read *)
          ADDRESS,                  (* the address to read them to *)
     VAR  CARDINAL                  (* the number read - 0 implies end *)
    );
  (* note: the read operation may produce device errors. *)
  (*       It is not a device error to attempt to read more data *)
  (*       than is available *)

  WriteProc =                       (* if canWrite, write data else termina
    PROCEDURE(
          Channel.Numbers,
          CARDINAL,                 (* number of storage units to write *)
          ADDRESS                   (* the address to write them from *)
    );
  (* note: the write operation may produce device errors *)

  ResetProc =                       (* if open, *)
    PROCEDURE(                      (* (re)initialise device instance *)
          Channel.Numbers           (* else terminate *)
    );
  (* note: the reset operation may produce device errors *)

  FlushProc =                       (* if canWrite, *)
    PROCEDURE(                      (* force buffered data to destination *
          Channel.Numbers           (* else terminate *)
    );
  (* note: the flush operation may produce device errors *)

  CloseProc =                       (* if open, *)
    PROCEDURE(                      (* close connection to device *)
          Channel.Numbers           (* else terminate *)
    );
  (* close connection from driver to device *)
  (* and break the link from the channel *)
  (* note: the close operation may produce device errors *)

  FreeProc =                        (* Tell driver that the channel is no *
    PROCEDURE(                      (* longer linked to it. *)
          Channel.Numbers
    );
  (* The driver shall carry out actions as CloseProc *)
  (* but shall not report any errors *)

  DriverProcs =
    RECORD
      doTest: TestProc;                     (* always legal *)
      doRead: ReadProc;                     (* terminates if not canRead *)
      doWrite: WriteProc;                   (* terminates if not canWrite *
      doReset: ResetProc;                   (* terminates if not open *)
      doFlush: FlushProc;                   (* terminates if not canWrite *
      doClose: CloseProc;                   (* terminates if not open *)
      doFree: FreeProc;                     (* always legal *)
    END;

  DriverRefs = POINTER TO DriverProcs;

(* a device driver module sets up a link between a numbered channel *)
(* and a table of driver procedures *)

PROCEDURE MakeLink(cn: Channel.Numbers; dr: DriverRefs);
(* pre : dr points to an initialised table of driver procedures *)
(* do:   if the channel is Open then terminate *)
(* post: the channel is Linked with the referenced table of driver *)
(*       procedures. *)
(* note: the driver module may subsequently change the fields of the *)
(*       driver table.  The table itself must remain in existence for *
(*       as long as the link exists *)

(* The following group of procedures allows the current drivers for a *
(* channel to be obtained.  If the channel is not Linked then *)
(* a procedure which terminates when called shall be returned *)

PROCEDURE ToTest(cn: Channel.Numbers): TestProc;
PROCEDURE ToRead(cn: Channel.Numbers): ReadProc;
PROCEDURE ToWrite(cn: Channel.Numbers): WriteProc;
PROCEDURE ToReset(cn: Channel.Numbers): ResetProc;
PROCEDURE ToFlush(cn: Channel.Numbers): FlushProc;
PROCEDURE ToClose(cn: Channel.Numbers): CloseProc;
PROCEDURE ToFlush(cn: Channel.Numbers): FlushProc;

PROCEDURE IsLinked(cn: Channel.Numbers): BOOLEAN;
(* post: returns TRUE iff the channel is Linked to a table of drivers *

(* When the device Close routine is called,
it is responsible for breaking the Link *)

PROCEDURE BreakLink(cn: Channel.Numbers);
(* post: the channel is not Linked with driver procedures. *)

(*
The module shall use IOTermination to establish a termination procedure
which will call  the Free procedure on all Linked channels.
If a device driver module needs to establish its own termination
procedure then this shall close its own open channels.
*)

END Drivers.
```

Will Modula-2 be Successful? NO!

John Lancaster

DigiSoft Design Ltd
90 Queens Rd, Feltham
TW13 5AR, England

I have been using Modula-2 for some time and watching the efforts of the BSI Modula-2 Working Group [1] to standardize the language. Throughout this period I have often asked myself:

> 'Will Modula-2 be efficient/versatile enough to be a winner in the general programming language market place?'

In the form proposed by Wirth I feel the answer is no. The BSI Working Group has done much to improve the situation and I would like to thank its members for their unpaid efforts. They have formalised the language definition and introduced extensions/-changes to it where necessary: multi-dimension open arrays[2] and co-routines [3] are two examples. However, there are still deficiencies. Below I highlight a number of problems and propose solutions to them.

PROBLEM 1. Structured Constants

At present structured constants are implemented by declaring a global variable which is initialized at runtime by code in the body of each module.

There are two disadvantages to this approach:

1. The "constant" is not safe because it is really a variable, hence the compiler cannot protect it from unintentional change.

2. For those structures whose value cannot be derived by simple computation, constants are duplicated in the data and code areas of the program. This can be a high overhead in memory-sensitive ROM based systems.

SOLUTION 1. Allow for the declaration of structured constants (CONST) as provided in the Turbo Pascal [4] dialect of Pascal.

PROBLEM 2. Parameter types.

Consider the following two procedures:

```
PROCEDURE MatrixOp (Op1, Op2 : ARRAY OF WORD,
                    VAR Result : ARRAY OF WORD)

PROCEDURE Length (String : ARRAY OF CHAR) : CARDINAL
```

These procedures have their input parameters passed by value. Although this is a safe method, the time spent creating a local copy of input parameters can have a severe effect on execution speed.

Many programmers consider the above inefficient and work around it by the unsafe practice of declaring structured input parameters as VARs. Unfortunately in the case of the 'Length' procedure this causes another problem, namely the following is not legal:

```
Size := Length('Literal String')
```

SOLUTION 2.Add to the language a new formal parameter PVAR (protected variable). The implementation of a PVAR parameter is identical to that of a VAR parameter except that compile-time checks protect it from modification within the procedure by not allowing assignment to it or its use as a VAR parameter.

As an aside I would also like to be able to code

```
Result := MatrixOP (Op1, Op2)
```

where the function is supplied with a pointer to Result and directly operated on it rather than creating an internal result which is passed out and assigned to Result. Although the syntax of Pascal could be modified to accommodate it, it appears that Modula-2's syntax cannot. Does anyone have any idea?

PROBLEM 3. User exception handling.

There is no provision in Modula-2 for the programmer to implement exception handling. This is primarily due to the absence of an equivalent to the Pascal construct "GOTO Label_InOuterBlock". The BSI proposed IO library [5] works around this shortcoming by predicate pretesting, i.e. testing if an operation can be performed before trying to do it. Although such an approach has merit it is not universally applicable.

SOLUTION 3. Allow for user-written exception handlers as in ADA. Borland's [6] have proposed a possible Modula-2 implementation.

PROBLEM 4. BITSET size and syntax.

The data type BITSET provides a simple mechanism for bit addressing (thereby allowing the crippled data type SET to be replaced by something more useful) and performing logical operations on word wide variables.

The deficiencies of BITSET become apparent on a machine with a variable word size addressing architecture like the 8086 & 68000 microprocessor families. Direct bit-twiddling of the hardware registers on such machines requires the language to support more than one size of BITSET. For the 68000 family BITSETs of 8, 16 & 32 elements (bits wide) are required.

SOLUTION 4. Introduce a new type construction, which needs to be imported from SYSTEM, with syntax of the form:

```
RegBitMap = BITSET OF [TxOn,RxOn,NIL,NIL,Reset,NIL,NIL,Error]
```

The size of the memory 'word' being addressed is given by the number of elements in the set. Allowing the elements of the set to belong to an enumerated type in addition to a CARDINAL sub-range brings to bit addressing the same benefits it gives to variable (word) addressing. The reserved word NIL is used as a padding (spacing) element, but also indicates to the compiler those bits which should not be accessed.

PROBLEM 5. Word subfields.

Consider a hardware device with the following bit allocation

```
---------------------------------
|. . . .|* * * *|* * . .|. . . .|
| | | | | | | | | | | | | | | |
---------------------------------
```

where '.' are 1-bit flags
and '*' is a 6-bit integer

To be efficient a low-level module which accesses the integer subfield should generate in-line code for bit shifting and sign extension.

SOLUTION 5. Require SYSTEM to export Shift and SignExtend functions which operate on all word sizes.

PROBLEM 6. Low-level escape path.

When SYSTEM does not export a suitable low-level facility the programmer must resort to calling an assembly language module. Such a solution is unattractive because:

1. Calling a module to execute one or two machine instructions is inefficient.

2. The special link format used by many of the currently available Modula-2 compilers cannot (easily) be linked with non-Modula-2 object files.

SOLUTION 6. SYSTEM must export a "code insert" facility.

I have presented above a number of extensions to the currently proposed BSI Modula-2 standard. With the exception of the BITSET proposal their adoption would not invalidate any existing code. I feel they are justified by the inability of the proposed language to implement these features with (library) modules else they are justified by the resulting increase in program reliability and efficiency they offer.

Although I welcome comment on my proposals I feel that the interests of the community are best served by submissions to the BSI Modula-2 Working Group and encourage readers to do so.

REFERENCES.

1. The Modula-2 Working Group of the British Standards Institution can be contacted via Barry Cornelius, Department of Computer Science, University of Durham, Durham, DH1 3LE, United Kingdom.
   Barry_Cornelius%mts.durham.ac.uk@UCL-CS.ARPA
   Barry_Cornelius@uk.ac.durham.mts
   bjc@uk.ac.nott.cs

2. "BSI Accepted Change: Multi-dimensional open arrays", Willy Steiger, "MODUS Quarterly" Issue 5, pp. 8-9.

3. "Coroutines and Processes", Roger Henry, "BSI Modula-2 Working Group, Second Open Meeting", July 24th 1986.

4. Turbo Pascal is a registered trademark of Borland International Inc.

5. "Draft BSI Standard I/O Library for Modula-2", Susan Eisenbach, "MODUS Quarterly" Issue 5, pp. 15-18.

6. "Proposal for a standard library and an Extension to Modula-2", Odersky, Sollich & Weisert of Borland International, "MODUS Quarterly" Issue 4, pp. 13-25.

MODULA2 USE IN URBAN TRANSPORTATION VITAL CONTROL SYSTEM

by Regis LARDENNOIS, R&D Manager,
MATRA TRANSPORT INTERELEC division, 53 rue du Cdt ROLLAND 93350 Le BOURGET FRANCE

INTRODUCTION

MATRA TRANSPORT is a company specialized in automatic transportation systems. It has realized the first worldwide metro with unmanned operation in LILLE (FRANCE) and is now is involved in many projects of train control vital functions with microprocessors. The first of them, the PARIS "A line" express metro will be in operation late 1987 with 200 trains. An other one with full automatic operation is scheduled in 1989 in LYON D line.
Realization of vital function uses very specific techniques, whatever may be the technology used, taking into account the high level of safety required and the need to ensure safety without maintenance operations.
The object of this paper is to describe the methods and tools used for the realisation of software into microprocessorized vital functions.

VITAL FUNCTIONS REALIZATION WITH MICROPROCESSORS

The object of this paper is not to describe the reasons for the use of microprocessors and methods of protection against failures during program execution, but for a good comprehension of the problems in software design, it is necessary to give some explanations.
The introduction of microprocessors for vital functions was decided regarding the great complexity of vital functions to be realized. This complexity is due to performance level requirements, to system adaptibility and to upward compatibility with existing signal protection systems.
It is also a means of reducing the cost of our train protection and control systems.
In urban transportation control system vital functions, the technological choice of several companies (FRENCH and US) has been to use single processors systems. The safety insurance is obtained by an extension of signature analysis concept. The reason for this choice is that the safety level is less dependant on maintenance tasks and of component technology than the redundancy between several processors concept.
The decision to use a high level language for application programming was also taken.
In this system, each data has a 32 bits machine representation and is associated with a 64 bits signature (or code). The signature of each data is representative of its identity, history (ancestors data), iteration number for loop variables and is modified by each operator modifying the data.
Each arithmetic or logic operator modifies the code in a different way in order to be able to verify the correct execution of operation. The basis of the signature realisation is an arithmetic code, enhanced to detect basic errors such as addressing errors, memory refresh errors, and to allow boolean variable/operators and tests.
Branches in the programs have the consequence that the same variables are not modified at each iteration, modified variables depending upon the result of branch tests. In order to obtain the same signatures associated with variables at the end of each microcomputer cycle, it is necessary to correct the signatures of unmodified variables at each convergent point between two branches.
This signature correction method needs the use of structured programming and is time-consuming for the CPU. It uses a data table at each convergent point into the program.
The correction operation is initialized by a procedure call at each such point, and the data table is created by a specific software tool realizing a dynamic simulation of program execution.
The signature processing is done with the objective that any error could be considered as a random modification of one or several variables, and the safety demonstration is realized by validation of this approximation with regard to physical failure mode types of the microcomputer.
The global verification of processing is done by testing the sequence of signatures of each output data, using well-known fail safe circuits.
A US signal company is using the same approach for microcomputer vital functions, but this company has developped a product comparable with a field programmable logic system for interlocking vital logic, whereas we were interested by including the most part of train control systems in the microprocessorized functions, with specific software, and developped a complete software development environment.

MICROCOMPUTER SYSTEM ARCHITECTURE

The microcomputer architecture used for vital functions is uses a 68000 microcomputer configuration. A 68020 version is under design.
The software environment for program execution includes 4 parts :
- A real time monitor,
- A vital operators basic environment providing about 120 elementary operators for processing both data and associated signatures,
- The 68000 code,
- Signatures data tables, specific to each application software.

The vital operators basic environment is written in assembly language, for time consuming optimisation, and all the other code, real time monitor and code specific to the application is written in Modula2.

REASONS OF MODULA2 CHOICE

Modula2 was choosen for several reasons :
- For industrial reasons, we wanted absolutely to control completely all software tools used for 68000 code generations. We had to develop a large volume of software, while using the facilities of the language to access the machine and to interface with assembler. It was not possible to accept in such industrial applications that the necessity to use a new release of a compiler (for instance because the older is not compatible with a new version of the development machine O.S.) involves incompatibilities with developed software. (We had the problem once with a 6809 PASCAL compiler which changed the way to pass parameters to procedures).
It is normal to accept evolution of software, but in industrial projects, the choice to use a new software tool version has to be a volontary decision of the software development manager.
I hope this point of view will be meditated by software tools developpers and that their products will take into account this industrial problem.
At the time we had to solve this problem (1983), it was easy to obtain the SMILERX compiler from ETH Züerich university.
This package is written in CDC PASCAL3 and we required 6 months of effort to adapt it to DIGITAL VAX PASCAL, this long work time being due to the poor portability of CDC PASCAL3 programs. In spite of this inconvenience, the quality of the software product is very good and we are very satisfied by the result.
We have with SMILERX and all its compiler and linker options a very powerful tool for developping industrial applications running with program in PROM.
Let us give some examples of available options :
<u>compiler options</u> - 16/32 bits words, stackTest/rangeCheck/no generate CLR in 68000 code (a CLR instruction generates a read/write cycle in order to position the state register and is inadequate in peripheral access);
<u>linker options</u> - ROMABLE code/generation of vectors/partial links;

- Modula2 gave us also all possibilities of a wealthy language, for real time or machine access needs, and the ability for several applications to forget completely assembly language. It should however be apreciated to have a real assembler inside the Modula2 development system, even if the assembler part in the software development is small.
- Modula2 gave us also the facility to test exactly the same code onto a mainframe environment (in our case, VAX VMS) with symbolic debugger, and a powerful O.S.. This a very important point.
We tested developments of programs in PASCAL language in a mainframe environment, and then transposed them onto microcomputers. But the number of modifications of the source program was so large that after modifications, it was not possible to test it again into the mainframe environment.
On the contrary with Modula2, it is possible to break own the software into modules allowing to compile and test the most part of the software in mainframe environment without any modification. Of course, input/output modules are different in the two environments, and we had to write mainframe modules emulating IO functions of the microcomputer environment.
I think that this is a major interest of Modula2 for developping microprocessor software and I hope that software developpers will enhance this type of application while designing their products. Let me suggest to them some ideas:
- Give different binary file prefix names into the two programming environments. Whereas the mainframe compiler and

the cross compiler have not necessarily the same designer, it should be better to give the user the facility to parameter file name suffix.
- Give the possibility to redefine the SYSTEM module name onto mainframe environment, and to implement specific procedures in a new SYSTEM module emulating completely the cross compiler SYSTEM module, with low-level machine access. This new module written by the user could also take into account the difference into machine representation for instructions such as SHIFT. For instance, bytes are not ordered in the same order into a word in a VAX and in a 68000.
- On 68000 microprocessors software, we generally prefer use 16 bits WORDS, to reduce run time and memory size. It should be very interesting to have the choice of 16 or 32 bits word size onto the mainframes compilers in order to simulate correctly microprocessor object programs. It should not be required to have a high level of optimisation for a 16 bits code. In any case, the efficiency of mainframe CPU use should be much greater than use of a microprocessor software simulator.
The first two suggestions are trivial and the third is generally partially implemented in mainframe compilers.

DEVELOPMENT CONFIGURATION

The configuration is described in drawing n° 1.
We use the Hamburg university VAX VMS Modula2 compiler for the test of all software on the mainframe configuration.
For the generation of 68000 code, we use our adaptation of SMILERX compiler/linker.
Onto the VAX computer, we introduced a Modula2 version of all the vital operators in order to be able to test completely the whole process.
The signature data table generator is an independant program processing all source program files and generating ROMABLE data.
We considered that software tests of modules is very important and organized these tests in order to be able to make archives of these tests and to be able to do easily non regression tests (using EXEC files for tests driving, and text files for results. Automatic comparison of text files results is easy.)
Microprocessor configuration tests are generally conducted in a first phase with a 68000 BSO software simulator, and then using an emulator.

SOFTWARE VALIDATION

By now, we have described the technological part of the realization. An other important part (probably the major part) is the software validation.
We need to be absolutely sure that the software don't contain any bug that could make vital functions protection inefficient.
This is made possible by the use of several methods :
- use of *Software Quality Plan*, defining the minimum rules to be applied into the development steps concerning the documentation, tests, registration of module versions at each step, non regression tests after modifications...
- software reviews during the software production,
- software test of each module,
- functional simulation of racks with simulator,
- extensive field tests,
- analysis of the software structure by modelization and simulation.

We tried to implement a partial formal validation of the software using Hoare Assertion proof, but we could'nt yet apply it despite the help of software tools developped to make easier the manipulation of formal assertions.

The principle of test by signature verification gives a very good protection against compiler errors, and we detected very early in the project all errors into our SMILERX compiler VAX adaptation.

I think that for this type of application,, software validation is the domain where there remains the most work to be done. The main difficulty is to make the relation between the functional requirements that generally are not very accurate (when applied to a large system) and the software specification and implementation.

About accuracy of functional requirements, remember that the train protection subsystem has to ensure safety into any situation that could happen to the trains, over many aspects. For many functions, functional requirements may be very accurate and detailed, but some times, there may be a contradiction between points of view and it is necessary to do trade offs. For instance, in case of smoke generation in a train, is it better to stop the train in a tunnel (with the risk of killing people by suffocation in the tunnel) or to drive the train as fast as possible to the next station (with the risk of killing people in the train if the time necessary to reach the station is too long, and also if the fire is too violent to envelop the station.
An other example is the interface between automatic systems and human intervention after an human error because to specify what to do after an human error could be interpreted as an acceptation of the possibility of such an error and decrease the level of responsability of human action in such systems.
Happily for passengers safety, human mistakes and fires are not frequent situations but it is absolutely necessary to know what to do in any situation.
In many situations, the best action is not obvious and the functional requirement is not accurate.

We have therefore to live with inaccurate functional specifications, generally with major difficulties to obtain detailed information about what to do in each case.
Two types of attitudes are possible :
- write a detailed functional implementation specification and ask for the agreement of the customer,
- rewrite the customer specification with a specification language such as a 5th generation language, and verify that the application software is never in contradiction with the known specification (in operation, or during an extensive test phase only).

The first approach is a good technical approach, but may be bad over a strictly contractual point of view because it induces a lot of modification on subjects that don't always merit the time spent to improve this system definition.
The second one is a more contractual approach.
The optimal solution is certainly a mixture of these two methods, and I think that software engineering methods that could interface classical software generation and 5th generation language execution should be very promising.

SOME FIGURES

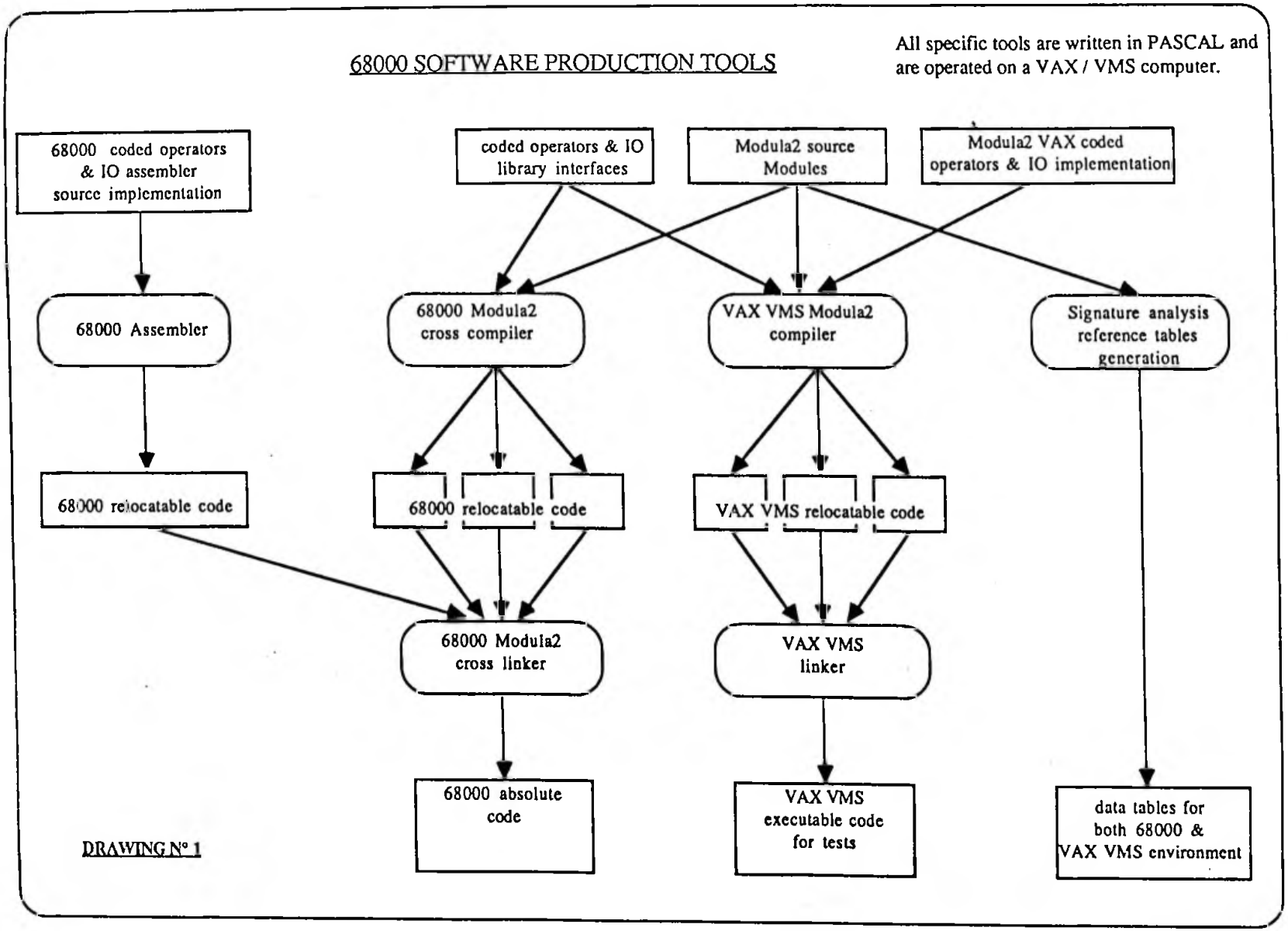
PARIS A line train control system has been realised jointly with two other FRENCH signal companies : Jeumont Schneider and C.S.E.E. The realized operational software for the 1st project is about 20000 source lines of Modula2 and 5000 source lines of assembler, plus about 20000 source lines of Modula2 and PASCAL software tools.
The total human effort is about 1000 man-months for system specification and software implementation. This is equivalent to 160000 hrs, or 6 hrs per source line of operational software. The total time necessary will have been 6 years between the beginning of detailed specification work and operation startup. This work was prepared by two years of technological feasibility studies and trade-offs.
Taking into account other projects in the development phase (which requires a much less effort because the 1st project supported almost all the developping effort), the team realising system specifications and software comprises 50 people.

CONCLUSION

Modula2 is practicable for large industrial software developments. We appreciate its ability to execute the same source module into several computer environments (including microprocessor boards), and also its well-known capabilities for machine access, modular development facilities and wealth of language capabilities.
I think that Modula2 qualities applied to industrial applications could be still further improved by compiler developpers.

ACKNOWLEDGEMENTS

Many thanks to Hermann SEILER of ETH Züerich and Dr Joachim SCHMITT of HAMBURG university for the good quality of their work, and for its distribution at low cost.


68000 SOFTWARE PRODUCTION TOOLS
All specific tools are written in PASCAL and are operated on a VAX / VMS computer.
68000 coded operators & IO assembler source implementation
coded operators & IO library interfaces
Modula2 source Modules
Modula2 VAX coded operators & IO implementation
68000 Assembler
68000 Modula2 cross compiler
VAX VMS Modula2 compiler
Signature analysis reference tables generation
68000 relocatable code
68000 relocatable code
VAX VMS relocatable code
68000 Modula2 cross linker
VAX VMS linker
68000 absolute code
VAX VMS executable code for tests
data tables for both 68000 & VAX VMS environment
DRAWING N° 1

Aron Felix Gurski
Strandlien 35
5000 Bergen
Norway

1986-10-27

Richard Karpinski
6521 Raymond Street
Oakland, CA 94609
USA

Dear Mr. Karpinski,

I enclose two copies of a short paper entitled "A Dhrystone Benchmark for PClones." I would like to submit this paper for publication in a future issue of "The MODUS Quarterly."

Should the paper be accepted and there be any difficulties with the hard copies, please let me know and I will gladly send the text on a diskette.

Sincerely,

A. Gurski

Aron Felix Gurski

A Dhrystone Benchmark for PClones
by Aron Felix Gurski

The benchmarks which have been used on PClones for the past years have by and large been programs written in Basic. Personally, I found the results of these benchmarks to be fairly useless inasmuch as I do not program in Basic. I wanted to be able to measure CPU speed based on the code generated by a compiler for the kind of language which I use.

Inasmuch as I have worked with mainframes for many years, I was acquainted with some of the benchmarks used in testing mainframes. One of these is the Dhrystone benchmark. For several years now, one of the standard benchmarks used in the mainframe world has been the one described by Reinhold P. Weicker in "Dhrystone: A Synthetic Systems Programming Benchmark" (*Comm. ACM*, vol. 27, no. 10, pp. 1013-1030). The program published in the article was written in Ada. As most members of MODUS are probably aware, Ada is sufficiently related to Modula-2 that a translation of many programs from Ada to Modula-2 is readily accomplished.

The Dhrystone program is based on 16 studies of the way programmers actually use the constructs available to them in various high-level languages. The languages studied include Ada, Algol 60, Algol 68, C, FORTRAN, Mesa, Pascal, PL/I, SAL and XPL. These are, with the exception of FORTRAN, all languages in the family to which Modula-2 also belongs: the Algol family. Hence, one may assume that the results of the statistical analyses of programs in these studies are also applicable to Modula-2.

Weicker has analyzed these studies carefully and written a program to reflect the distributions of constructs which they found. The resulting Dhrystone program is intended to be an "average" program biased towards systems programming. This means that there are no floating point calculations in the program; the Whetstone benchmark already exists to test the speed of floating point operations. However, a large portion of the constructs available in Modula-2 are used -- and used in varying combinations. A certain percentage of the variables are local, others are global. Of the global variables, some are imported from another module while others are declared in the module in which they are used. Some of the IF statements have an ELSE clause, some do not. Of those which do have an ELSE clause, some have an ELSE clause which is executed while others do not. And so on. There is no need for me to repeat information which has already been published; the interested reader is referred to Weicker's paper. Suffice it to say that the distribution has become an accepted norm for existing high-level languages.

In translating the program into Modula-2, I have made 4 types of changes: required syntactic changes, stylistic syntactic changes, cosmetic changes and timing changes. There were two required syntactic changes (aside from the obvious ones because the program was originally written in Ada): a reference to an uninitialized variable was replaced by a reference to an initialized variable, and a procedure was added to the program. Adding the procedure was necessary because Ada allows operator overloading, whereas Modula-2 does not. To compensate for this, a small procedure to compare two strings was necessary. The stylistic syntactic changes are the replacement of statements of the form

```
variable := variable + constant
```

by statements of the form

```
INC(variable, constant)
```

or

```
INC(variable).
```

The cosmetic changes are changes to the identifiers used in the program; they are more descriptive in the Modula-2 version. The timing changes are in the main program. In order to accomodate the speed of the PClones available today, the main program calls the Dhrystone procedure 30000 times. As faster machines become available, this number will have to be increased in order to give results which can be measured.

The Modula-2 version of the Dhrystone program seems to be portable -- a desirable feature for virtually any piece of software. The version which I have made available for testing PClones was compiled by the Logitech Modula-2/86 compiler. No options were used to inhibit code-generation for various run-time error conditions. For the sake of portability, no options were used to specify that code be generated for the Intel 80286 processor. The resulting code has run on quite a variety of computers from different manufacturers, some of which are known to be only minimally compatible with the "industry standard".

Appendix 1 contains the source code of the Dhrystone program in Modula-2. Appendix 2 is a list of the results which I have obtained so far. When testing PClones, I have always run the benchmark from diskette -- even on those machines which have a hard disk available. Wherever possible, I have used the version of DOS which was supplied by the manufacturer.

I would like to thank those firms which allowed me to test computers which they sell. Understandably, not all of them were equally pleased with the results I obtained. I would also like to thank Helge Vindenes, the operator of the Costa del Vindenes bulletin board system. The Dhrystone benchmark is posted there, and is available for downloading. The telephone number to Costa del Vindenes is +47 5 15 16 10 and the transmission rates are 300, 1200 and 2400 baud (using CCITT standard frequencies). The benchmark is also posted in IBMSIG on The Source.

Appendix 1

```
(*
    DHRYSTONE

    Author
        Aron Felix Gurski

    Date
        1986-07-27

    Purpose
        DHRYSTONE is a benchmark program which can be used to compare compute:
        performance

    Usage
        DHRYSTONE

    Options
        None

    Default values of the options
        None

    Remarks
        1. The Dhrystone benchmark reflects a "representative" distribution o:
           statement types, based on statistical analyses of programs. The
           approximate distribution is:
               assignment statements - 53%
               control statements    - 32%
               procedure calls       - 15%
        2. The program is balanced with respect to
               a. statement type,
               b. operand type (for simple data types), and
               c. operand access (global, local, parameter or constant)
        3. The program does not compute anything meaningful.
        4. This program is based on the version in Ada, published in "Dhrystor
           A Synthetic Systems Programming Benchmark" by Reinhold P. Weicker,
           Communications of the ACM, vol. 27, no. 10, pp. 1013 - 1030.
*)

MODULE Dhrystone;

IMPORT Module1;

CONST
    NoOfRepetitions = 30000;

VAR
    Count : CARDINAL;

BEGIN
FOR Count := 1 TO NoOfRepetitions DO
    Module1.Procedure0
    END (* FOR *)
END Dhrystone.
```

```
DEFINITION MODULE GlobalDefinitions;

EXPORT QUALIFIED CapitalLetter, Enumeration, Matrix, NoughtToTwentyNine,
                 OneToFifty, RecordPointer, RecordType, String30, Vector;

TYPE
    NoughtToTwentyNine = [0..29];
    OneToFifty = [1..50];

    CapitalLetter = CHAR;
    Enumeration = (Value1, Value2, Value3, Value4, Value5);
    Matrix = ARRAY OneToFifty, OneToFifty OF INTEGER;
    String30 = ARRAY NoughtToTwentyNine OF CHAR;
    Vector = ARRAY OneToFifty OF INTEGER;

    RecordPointer = POINTER TO RecordType;
    RecordType = RECORD
                     PointerComponent : RecordPointer;
                     CASE TagField : Enumeration OF
                         Value1:                    (* only this variant is used *
                             EnumerationComponent : Enumeration;
                             IntegerComponent : OneToFifty;
                             StringComponent : String30!
                         Value2:
                             EnumerationComponent2 : Enumeration;
                             StringComponent2 : String30
                         ELSE
                             CharComponent1 : CHAR;
                             CharComponent2 : CHAR
                         END (* CASE *)
                     END (* RECORD *);

END GlobalDefinitions.

IMPLEMENTATION MODULE GlobalDefinitions;

END GlobalDefinitions.


DEFINITION MODULE Module1;

FROM GlobalDefinitions IMPORT OneToFifty, RecordPointer;

EXPORT QUALIFIED GlobalInteger, Procedure0, Procedure1, Procedure2, Procedure3;

VAR
    GlobalInteger : INTEGER;

PROCEDURE Procedure0;

PROCEDURE Procedure1 (PointerInputParameter : RecordPointer);

PROCEDURE Procedure2 (VAR IntegerInputOutputParameter : OneToFifty);

PROCEDURE Procedure3 (VAR PointerOutputParameter : RecordPointer);

END Module1.
```

```
IMPLEMENTATION MODULE Module1;

FROM GlobalDefinitions IMPORT Enumeration, Matrix, OneToFifty, RecordPointer,
                              String30, Vector;
IMPORT Module2;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;

VAR
    GlobalBoolean : BOOLEAN;
    GlobalChar1 : CHAR;
    GlobalChar2 : CHAR;
    GlobalMatrix : Matrix;
    GlobalNextPointer : RecordPointer;
    GlobalPointer : RecordPointer;
    GlobalVector : Vector;

PROCEDURE Procedure0;

VAR
    CharIndex : CHAR;
    LocalChar : CHAR;
    LocalEnumeration : Enumeration;
    LocalInteger1 : OneToFifty;
    LocalInteger2 : OneToFifty;
    LocalInteger3 : OneToFifty;
    LocalString1 : String30;
    LocalString2 : String30;

BEGIN
NEW(GlobalNextPointer, Value1);
NEW(GlobalPointer, Value1);
WITH GlobalPointer^ DO
    PointerComponent := GlobalNextPointer;
    TagField := Value1;
    EnumerationComponent := Value3;
    IntegerComponent := 40;
    StringComponent := "DHRYSTONE PROGRAM, SOME STRING"
    END (* WITH *);
LocalString1 := "DHRYSTONE PROGRAM, 1'ST STRING";
Procedure5;
Procedure4;
LocalInteger1 := 2;
LocalInteger2 := 3;
LocalString2 := "DHRYSTONE PROGRAM, 2'ND STRING";
LocalEnumeration := Value2;
GlobalBoolean := NOT Module2.Function2(LocalString1, LocalString2);
WHILE LocalInteger1 < LocalInteger2 DO
    LocalInteger3 := 5*LocalInteger1 - LocalInteger2;
    Module2.Procedure7(LocalInteger1, LocalInteger2, LocalInteger3);
    INC(LocalInteger1)
    END (* WHILE *);
Module2.Procedure8(GlobalVector, GlobalMatrix, LocalInteger1, LocalInteger3);
Procedure1(GlobalPointer);
FOR CharIndex := "A" TO GlobalChar2 DO
    IF LocalEnumeration = Module2.Function1(CharIndex, "C") THEN
        Module2.Procedure6(Value1, LocalEnumeration)
        END (* IF *)
    END (* FOR *);
LocalInteger3 := LocalInteger2*LocalInteger1;
LocalInteger2 := LocalInteger3 DIV LocalInteger1;
LocalInteger2 := 7*(LocalInteger3 - LocalInteger2) - LocalInteger1;
Procedure2(LocalInteger1);
DISPOSE(GlobalNextPointer, Value1);
DISPOSE(GlobalPointer, Value1)
END Procedure0;
```

```
PROCEDURE Procedure1 (PointerInputParam : RecordPointer);

BEGIN
PointerInputParam^.PointerComponent^ := GlobalPointer^;
PointerInputParam^.IntegerComponent := 5;
PointerInputParam^.PointerComponent^.IntegerComponent :=
     PointerInputParam^.IntegerComponent;
PointerInputParam^.PointerComponent^.PointerComponent :=
    PointerInputParam^.PointerComponent;
Procedure3(PointerInputParam^.PointerComponent^.PointerComponent);
IF PointerInputParam^.PointerComponent^.TagField = Value1 THEN
     PointerInputParam^.PointerComponent^.IntegerComponent := 6;
     Module2.Procedure6(PointerInputParam^.EnumerationComponent,
         PointerInputParam^.PointerComponent^.EnumerationComponent);
     PointerInputParam^.PointerComponent^.PointerComponent :=
         GlobalPointer^.PointerComponent;
     Module2.Procedure7(PointerInputParam^.PointerComponent^.IntegerComponent,
         10, PointerInputParam^.PointerComponent^.IntegerComponent)
 ELSE
     PointerInputParam^ := PointerInputParam^.PointerComponent^
     END (* IF *)
 END Procedure1;

 PROCEDURE Procedure2 (VAR IntegerInputOutputParam : OneToFifty);

 VAR
     LocalEnumeration : Enumeration;
     LocalInteger : OneToFifty;

 BEGIN
 LocalInteger := IntegerInputOutputParam + 10;
 REPEAT
     IF GlobalChar1 = "A" THEN
         DEC(LocalInteger);
         IntegerInputOutputParam := LocalInteger - OneToFifty(GlobalInteger);
         LocalEnumeration := Value1
         END (* IF *)
     UNTIL LocalEnumeration = Value1
 END Procedure2;

 PROCEDURE Procedure3 (VAR PointerOutputParam : RecordPointer);

 BEGIN
 IF GlobalPointer # NIL THEN
     PointerOutputParam := GlobalPointer^.PointerComponent
 ELSE
     GlobalInteger := 100
     END (* IF *);
 Module2.Procedure7(10, GlobalInteger, GlobalPointer^.IntegerComponent)
 END Procedure3;

 PROCEDURE Procedure4;

 VAR
     LocalBoolean : BOOLEAN;

 BEGIN
 LocalBoolean := GlobalChar1 = "A";
 LocalBoolean := LocalBoolean OR GlobalBoolean;
 GlobalChar2 := "B"
 END Procedure4;

 PROCEDURE Procedure5;
```



```
 BEGIN
```

```
GlobalChar1 := "A";
GlobalBoolean := FALSE
END Procedure5;

END Module1.
```

```
DEFINITION MODULE Module2;

FROM GlobalDefinitions IMPORT CapitalLetter, Enumeration, Matrix, OneToFifty,
                              String30, Vector;

EXPORT QUALIFIED Function1, Function2, Procedure6, Procedure7, Procedure8;

PROCEDURE Procedure6 (EnumerationPar : Enumeration;
                      VAR EnumerationOutputParam : Enumeration);

PROCEDURE Procedure7 (IntegerInputParam1 : OneToFifty;
                      IntegerInputParam2 : OneToFifty;
                      VAR IntegerOutputParam : OneToFifty);

PROCEDURE Procedure8 (VAR VectorInputOutputParam : Vector;
                      VAR MatrixInputOutputParam : Matrix;
                      IntegerInputParam1 : INTEGER;
                      IntegerInputParam2 : INTEGER);

PROCEDURE Function1 (CharInputParam1 : CapitalLetter;
                     CharInputParam2 : CapitalLetter) : Enumeration;

PROCEDURE Function2 (StringInputParam1 : String30;
                     StringInputParam2 : String30) : BOOLEAN;

END Module2.
```

```
IMPLEMENTATION MODULE Module2;

FROM GlobalDefinitions IMPORT CapitalLetter, Enumeration, Matrix,
                              NoughtToTwentyNine, OneToFifty, String30, Vector;
IMPORT Module1;

PROCEDURE Procedure6 (EnumerationInputParam : Enumeration;
                      VAR EnumerationOutputParam : Enumeration);

BEGIN
EnumerationOutputParam := EnumerationInputParam;
IF NOT Function3(EnumerationInputParam) THEN
    EnumerationOutputParam := Value4
    END (* IF *);
CASE EnumerationInputParam OF
    Value1:
        EnumerationOutputParam := Value1|
    Value2:
        IF Module1.GlobalInteger > 100 THEN
            EnumerationOutputParam := Value1
        ELSE
            EnumerationOutputParam := Value4
            END (* IF *)|
    Value3:
        EnumerationOutputParam := Value2|
    Value4:
        |
    Value5:
        EnumerationOutputParam := Value3
    END (* CASE *)
END Procedure6;

PROCEDURE Procedure7 (IntegerInputParam1 : OneToFifty;
                      IntegerInputParam2 : OneToFifty;
                      VAR IntegerOutputParam : OneToFifty);

VAR
    LocalInteger : OneToFifty;

BEGIN
LocalInteger := IntegerInputParam1 + 2;
IntegerOutputParam := IntegerInputParam2 + LocalInteger
END Procedure7;

PROCEDURE Procedure8 (VAR VectorInputOutputParam : Vector;
                      VAR MatrixInputOutputParam : Matrix;
                      IntegerInputParam1 : INTEGER;
                      IntegerInputParam2 : INTEGER);

VAR
    IntegerIndex : INTEGER;
    LocalInteger : OneToFifty;

BEGIN
LocalInteger := IntegerInputParam1 + 5;
VectorInputOutputParam[LocalInteger] := IntegerInputParam2;
VectorInputOutputParam[LocalInteger + 1] :=
    VectorInputOutputParam[LocalInteger];
VectorInputOutputParam[LocalInteger + 30] := LocalInteger;
FOR IntegerIndex := LocalInteger TO LocalInteger + 1 DO
    MatrixInputOutputParam[LocalInteger, IntegerIndex] := LocalInteger
    END (* FOR *);
```



```
(*
    I have changed the index in the following statement because it origin.
```

```
    referred to an uninitialized element in MatrixInputOutputParameter; the
    original program referred to
        MatrixInputOutputParam[LocalInteger, LocalInteger - 1].
*)
INC(MatrixInputOutputParam[LocalInteger, LocalInteger + 1]);

MatrixInputOutputParam[LocalInteger + 20, LocalInteger] :=
    VectorInputOutputParam[LocalInteger];
Module1.GlobalInteger := 5
END Procedure8;

PROCEDURE Function1 (CharInputParam1 : CapitalLetter;
                     CharInputParam2 : CapitalLetter) : Enumeration;

VAR
    LocalChar1 : CapitalLetter;
    LocalChar2 : CapitalLetter;

BEGIN
LocalChar1 := CharInputParam1;
LocalChar2 := LocalChar1;
IF LocalChar2 # CharInputParam2 THEN
    RETURN Value1
ELSE
    RETURN Value2
    END (* IF *)
END Function1;

PROCEDURE Function2 (StringInputParam1 : String30;
                     StringInputParam2 : String30) : BOOLEAN;

VAR
    LocalChar : CapitalLetter;
    LocalInteger : NoughtToTwentyNine;

(*
    I have added Comparison to Function2. The original program compared the tw
    parameters of Function2 by using overloading of the ">" operator in Ada;
    Modula-2 has no operator overloading, so a PROCEDURE is necessary to perfo
    the same function.
*)

PROCEDURE Comparison (VAR StringInputParam1 : String30;
                      VAR StringInputParam2 : String30) : INTEGER;

VAR
    Index : INTEGER;
    State : (Scanning, Different, Same);

BEGIN
Index := -1;
State := Scanning;
WHILE (State = Scanning) AND (Index < 29) DO
    INC(Index);
    IF StringInputParam1[Index] # StringInputParam2[Index] THEN
        State := Different
    ELSIF Index = 29 THEN
        State := Same
        END (* IF *)
    END (* WHILE *);
CASE State OF
    Same:
        RETURN 0;
    Different:
        IF StringInputParam1[Index] < StringInputParam2[Index] THEN
            RETURN -1
```

```
        ELSE
            RETURN -1
            END (* IF *)
    END (* CASE *)
END Comparison;

BEGIN
LocalInteger := 2;
WHILE LocalInteger <= 2 DO
    IF Function1(StringInputParam1[LocalInteger],
        StringInputParam2[LocalInteger + 1]) = Value1 THEN
        LocalChar := "A";
        INC(LocalInteger)
        END (* IF *)
    END (* WHILE *);
 IF (LocalChar >= "W") AND (LocalChar < "Z") THEN
    LocalInteger := 7
    END (* IF *);
 IF LocalChar = "X" THEN
    RETURN TRUE

 (*
    I have modified the following ELSIF clause; the original program used Ada's
    operator overloading to compare the two strings.
 *)
 ELSIF Comparison(StringInputParam1, StringInputParam2) > 0 THEN

    INC(LocalInteger, 7)
 ELSE
    RETURN FALSE
    END (* IF *)
 END Function2;

 PROCEDURE Function3 (EnumerationInputParam : Enumeration) : BOOLEAN;

 VAR
    LocalEnumeration : Enumeration;

 BEGIN
 LocalEnumeration := EnumerationInputParam;
 IF LocalEnumeration = Value3 THEN
    RETURN TRUE
    END (* IF *)
 END Function3;

 END Module2.
```

Appendix 2

Machine	Operating system	Dhrystones per second
Bull Micral 30	MS-DOS 3.10	50.8
Bull Micral 60 (8 MHz)	MS-DOS 3.10	199.0
Commodore PC 20-II	MS-DOS 2.11	50.8
Compaq Plus	PC-DOS 2.10	51.0
Compaq Portable II	MS-DOS 3.10	184.2
Ericsson Portable PC	PC-DOS 3.10	51.0
Ericsson Workstation 286	MS-DOS 3.10	261.3
Hewlett Packard Vectra	MS-DOS 3.10	203.2
IBM 3270 AT	PC-DOS 3.10	147.0
IBM 3270 PC	PC-DOS 3.10	51.0
IBM PC	PC-DOS 2.10	51.0
IBM PC/AT (6 MHz)	PC-DOS 3.10	100.3
IBM PC/XT	PC-DOS 2.10	51.0
Kaypro 286i	MS-DOS 3.20	197.5
Kaypro 286i	PC-DOS 3.10	197.5
Multitech Plus 700 (4.77 MHz)	MS-DOS 2.11	50.8
Multitech Plus 700 (8 MHz)	MS-DOS 2.11	84.2
NCR PC6 (4.77 MHz)	MS-DOS 2.11	50.8
NCR PC6 (8 MHz)	MS-DOS 2.11	83.0
NEC APC III	MS-DOS 2.11	104.2
Olivetti M19	PC-DOS 3.10	50.9
Olivetti M24	MS-DOS 3.10	111.2
Olivetti M28	MS-DOS 3.10	199.2
Philips Flyer	MS-DOS 3.10	58.9
Philips P3102	MS-DOS 3.10	55.5
Philips P3200	MS-DOS 3.10	156.7
Sanyo MBC-990 (6 MHz)	PC-DOS 3.10	144.8
Sanyo MBC-990 (8 MHz)	PC-DOS 3.10	196.8
Tandon AT	MS-DOS 3.10	144.7
Tandon AT with AT Gizmo CPU	PC-DOS 3.20	199.0
Tandon XT	MS-DOS 2.11	43.1
Tandy 1000 EX	PC-DOS 3.10	45.9
Tandy 3000 HL	MS-DOS 2.11	198.8
Toshiba T1100	MS-DOS 2.11	49.2
Toshiba T2100 (4.77 MHz)	PC-DOS 3.10	69.4
Toshiba T2100 (7.16 MHz)	PC-DOS 3.10	103.0
Toshiba T3100 (6 MHz)	MS-DOS 2.11	94.0
Toshiba T3100 (8 MHz)	MS-DOS 2.11	232.6
Wang PC	MS-DOS 2.01	116.0

Modula-2 News Issue # 0 October 1984

Purposes, practices and promises for Modula-2 News
Revisions and Amendments to Modula-2, Niklaus Wirth
Specification of Standard Modules, Jirka Hoppe
Modula-2 in the Public Eye (a bibliography), Winsor Brown
Modus Membership list, by name
Modus members's addresses, by location
Modula-2 Implementation Questionnaire

Modula-2 News Issue # 1 January 1985

Review of Gleaves' Modula-2 text by Tom DeMarco
MODUS Paris meeting 20/21 Sep 84, C.A. Blunsdon
Report of M2 Working Group, 8 Nov 84, John Souter
Modula-2 Standard Library Rationale, Randy Bush
Modula-2 Standard Library Definition Modules
Modula-2 Standard Library Documentation, Jon Bondy
Validation of M2 Language Implementations, J. Siegel

MODUS Quarterly # 2 April 1985

Letters, Anderson & Emerson
Opaque Types in Modula-2, C. French & R. Mitchell
Dynamic Module Instantiation, Roger Sumner
The Linking Process in Modula-2, Jeanette Symons
Modula-2 Library Comments, Bob Peterson
Modula Compilers - Where to Get 'em, Larry Smith
Coding War Games Prospectus, Tom DeMarco
M2, An Alternative to C, M. Djavaheri, S. Osborne

MODUS Quarterly # 3 July 1985

Letters, Endicott & Hoffman
Some Thoughts on Modula-2 in "Real Time", Paul Barrow
RajaInOut: simple, safer, I/O for
Logitech/MS-DOS, R. Thiagarajan
Selection of Contentious Problems, Barry Cornelius
Expressions in Modula-2, Brian Wichmann
The Scope Problems Caused by Modules, Barry Cornelius

MODUS Quarterly # 4 November 1985

State of MODUS, George Symons
MODUS Meeting Report, Bob Peterson
A Writer's View of a Programmer's Conference, Sam'l Bassett
Concerns of A programmer, Dennis Cohen
Modifications to the Standard Library
Proposal, R. Nagler & J. Siegel
Proposal, standard library and M2 extension,
Odersky, Sollich, & Weisert
Standard Library of the Unix OS, Morris Djavaheri
The Standard Library for PC's, E. Verhulst
Editorial, Richard Karpinski
Modula-2 Compilation and Beyond, D.G. Foster
Modula-2 Processes - Problems and Suggestions, Roger Henery

MODUS Quarterly # 5 February 1986

Editorial, Richard Karpinski
Exporting a Module Identifier, Barry Cornelius
Letter on multi dimensional open arrays, Niklaus Wirth
Letter on DIV, MOD, /, and REM, Niklaus Wirth
BSI Accepted Change: Multi-dim. open arrays, Willy Steiger
N73: NULL-terminated strings in Modula-2, Ole Poulsen
ISO Ballot Results re BSI Specifying Modula-2
Draft BSI Standard I/O Library for Modula-2, Susan Eisenbach
Portable Language Implementation Project: Design and
Development Rationale, K. Hopper and W.J. Rogers
The ETH-Zuerich Modula-2 for the Macintosh, Chris Jewell
NewStudio: Engineering a Modula-2 Application for the Mac,
A. Davidson, H.B. Herrmann, E.R. Hoffer

MODUS Quarterly # 6 November 1986

Editorial, Richard Karpinski
Letter on opaque types, File type, and SET OF CHAR, P. Willia
Letter on exported identifiers, E. Videki
Why the Plain Vanilla Linkers, J. Gough
Letter re best article & MacModula-2, M. Coren
Significant Changes to the Language Modula-2, Barry Cornelius
All About Strings, Barry Cornelius
Type Conversions in Modula-2, B. Wichmann
Improving the quality of Definition Modules, A. Sale
A Programming Environment for Modula-2, F. Odegard
Academic Modula-2 Survey, L. Mazlack
Compilers for Modula-2 (Zuerich list)
Membership List

MODUS Quarterly # 7 February 1987

Editorial, Richard Karpinski
New Products
Modula-2 Standardisation: A go betweens tale, Welsh & Bailes
Modula-2 VM/CMS, Thomas Habernoll
TCP Implementation in Modula-2, F. Ma & L. D. Wittie
Building an Operating System with Modula-2,
B. Justice, S. Osborne, & V. Wills
Note on Implementing SET OF CHAR, Source Code
for a SetOfChar MODULE, A. Brunnschweiler

MODUS Quarterly # 8 May 1987

Editorial, Richard Karpinski
Letter re unwarranted BSI changes, T. DeMarco
Response to DeMarco letter, R. Karpinski
Letter re standards questions, A. R. Spitzer
Open Letter from a Practicing Programmer, W. Nicholls
Coroutines and Processes, R. Henery
Another look at the FOR statement, B. Cornelius
Automatic export of identifiers from the definition module,
A. H. J. Sale
BSI Modula-2 Working Group Standard Concurrent Programming
Facilities, D. Ward

The above back issues are still in print. MODUS Administrators supply single copies at $5 US or 12 Swiss Francs.

Modula-2 Users' Association
MEMBERSHIP APPLICATION

Name : _____

Affiliation : _____

Address : _____

Address : _____

City : _____

State : _____ **Postal Code:** _____ **Country:** _____

Phone : (___)____-_____ Electronic Addr : _____

Application as: New Member _____ **or Renewal** _____

Implementation(s) used : _____

Option: ___ Do NOT print my phone number in any rosters
or: ___ Print ONLY my name and country in any rosters
or: ___ Do NOT release my name on mailing lists

** Membership fee per year (20 USD or 45 SFr) **
Members of the US group who are outside of North America, add $10.00.

In North and South America, please send check or money order (drawn in US dollars) payable to Modula-2 Users' Association at:

Modula-2 Users' Association
P.O. Box 51778
Palo Alto, California 94303
United States of America

Otherwise, please send check or bank transfer (in Swiss Francs) payable to Modula-2 Users' Association at:

Aline Sigrist
MODUS Secretary
ERDIS SA
Postfach 35, CH-1800 Vevey 2
Switzerland

The Modula-2 Users' Association is a forum for all parties interested in the Modula-2 Language to meet each other and exchange ideas. The primary means of communication is through the Newsletter which is published four times a year. Membership is for an academic year, and you will receive all newsletters for the full year in which you join. Mid-year applications receive that year's back issues. Modula-2 is a new and developing language; this organization provides implementors and serious users a means to discuss and keep informed about the standardization effort, while discussing implementation ideas and peculiarities. For the recreational user, there is information on the status of the language, along with examples and ideas for programming in Modula-2. For everyone, there is information on current implementations and the other resources available for obtaining information on the language.

Modula-2 Users' Association
MEMBERSHIP APPLICATION

Name : _____

Affiliation : _____

Address : _____

Address : _____

City : _____

State : _____ **Postal Code:** _____ **Country:** _____

Phone : (___)____-_____ Electronic Addr : _____

Application as: New Member _____ **or Renewal** _____

Implementation(s) used : _____

Option: ___ Do NOT print my phone number in any rosters
or: ___ Print ONLY my name and country in any rosters
or: ___ Do NOT release my name on mailing lists

** Membership fee per year (20 USD or 45 SFr) **
Members of the US group who are outside of North America, add $10.00.

In North and South America, please send check or money order (drawn in US dollars) payable to Modula-2 Users' Association at:

Modula-2 Users' Association
P.O. Box 51778
Palo Alto, California 94303
United States of America

Otherwise, please send check or bank transfer (in Swiss Francs) payable to Modula-2 Users' Association at:

Aline Sigrist
MODUS Secretary
ERDIS SA
Postfach 35, CH-1800 Vevey 2
Switzerland

The Modula-2 Users' Association is a forum for all parties interested in the Modula-2 Language to meet each other and exchange ideas. The primary means of communication is through the Newsletter which is published four times a year. Membership is for an academic year, and you will receive all newsletters for the full year in which you join. Mid-year applications receive that year's back issues. Modula-2 is a new and developing language; this organization provides implementors and serious users a means to discuss and keep informed about the standardization effort, while discussing implementation ideas and peculiarities. For the recreational user, there is information on the status of the language, along with examples and ideas for programming in Modula-2. For everyone, there is information on current implementations and the other resources available for obtaining information on the language.

MODUS
c/o Pacific Systems
P.O.Box 51778
US-Palo Alto, CA 94303

MODUS
c/o ER[illegible]S SA
Case Postale 35
CH-1800 Vevey 2