Portability Techniques for BLISS Programs

Bruce W. Leverett
Paul J. Knueven
Peter G. Hibbard

November, 1977

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

Approved for public release; distribution unlimited.
The problems of writing large systems in BLISS that can be run in more than one environment are described. A method (a set of methods) for attacking these problems is explained, with examples of its use in a particular system (a compiler). Aspects of the BLISS language are discussed with regard to their usefulness or uselessness in solving these problems.
Portability Techniques for BLISS Programs

Bruce W. Leverett
Paul J. Knueven
Peter G. Hibbard

November, 1977

Abstract

The problems of writing large systems in BLISS that can be run in more than one environment are described. A method (a set of methods) for attacking these problems is explained, with examples of its use in a particular system (a compiler). Aspects of the BLISS language are discussed with regard to their usefulness or uselessness in solving these problems.

The work described here was supported by the Defense Advanced Research Projects Agency under contract F44620-73-C-0074, monitored by the Air Force Office of Scientific Research.
In 1974, the authors undertook to write a compiler for ALGOL 68. The compiler was to run on a PDP-11 and produce code suitable for a PDP-11. The first version was to run on C.mmp[1], a multiprocessor system comprising several (slightly modified) PDP-11 processors and other hardware, under the HYDRA[2] operating system.

The compiler was written in BLISS[3], for the usual variety of reasons, not least of which was that the C.mmp/HYDRA system offers a complete symbolic debugging system (SIX12[4]) and other support for BLISS programs. However, there were some initial problems with building a system in this environment.

- The environment itself was unstable. The hardware and the various levels of the operating system were full of bugs, and subject to frequent redesign as well.

- The environment was completely unfamiliar to the authors, who had never used it before.

- The compiler and linker for BLISS did not run on C.mmp/HYDRA, but on a PDP-10/TOPS-10, with no easily available link between the two. Thus changing the compiler involved loading a linked version onto a DECtape, carrying the tape over to C.mmp, mounting it there, and reading it—a time-consuming process. (This, incidentally, was one of the problems that the ALGOL 68 system was intended to solve, since it was to run entirely under C.mmp/HYDRA.)

Hence, the authors decided to build a preliminary version of the compiler, which would run entirely under PDP-10/TOPS-10 (suffering from none of the above three problems), and do most of the early development work on the preliminary version. To minimize the problems of changing over from PDP-10 versions to C.mmp versions and vice versa, the preliminary version was also written in BLISS. We wished to be able to switch from using either version to using the other, simply by making a small number of error-proof changes, and then running the appropriate compiler and linker to produce the version to be used. We completely met this goal, and in fact switched from using one version to using the other several times. In addition, we met it in a way that was very easy to maintain: although we had not used the PDP-10 version since April 1976, we used the procedure outlined above to create a PDP-10 version in February 1977, with essentially no problems (e.g. all files compiled correctly, in spite of language differences between BLISS-10 and BLISS-11).

Achieving this level of portability was a non-trivial task. BLISS is not as high-level a language as ALGOL nor as standardized as FORTRAN (though achieving portability even in FORTRAN is not always easy [5]). The machine-dependence of the design of even the simplest data structures is not hidden from the programmer; moreover, I/O and other interactions with the operating system are not part of the language, and must be supplied by the user's program (or at least in separately-compiled programs). Standard packages are available for handling routine problems for both BLISS-10/TOPS-10 and BLISS-11/HYDRA, but there are several problems with using these:

- They are not compatible with each other.
- They (especially the BLISS-11/HYDRA package) are inadequate for the needs of large programs such as compilers.
There are no packages for any PDP-11 operating systems other than HYDRA, RSX-11, and RT-11, and even these three are not compatible with each other.

This paper describes the techniques we used to achieve portability, i.e., to minimize the work necessary to change over from one version to the other. Chapter 1 describes our system for isolating the differences between the architectures of the PDP-10 and the PDP-11. Chapter 2 describes our treatment of the differences between the BLISS-10 and BLISS-11 languages. Chapter 3 describes the compiler interface with the operating system, i.e., the system for isolating the differences between TOPS-10 and HYDRA. In Chapter 4 we draw some conclusions about the general applicability of our techniques, and about the design of system implementation languages (such as BLISS) with portability in mind. Portions of the compiler are included in the Appendices for illustration.

1. Hardware Differences

By using a high-level language, we were able to avoid having to deal with differences between the instruction sets and instruction formats of the PDP-10 and the PDP-11 (the few exceptions are noted below). However, because of the nature of the BLISS language, we could not avoid dealing with differences between their data formats.

Our method of isolating these differences was to use a "REQUIRE file". BLISS has a declaration similar in purpose to the PL/I INCLUDE statement, by means of which a program may incorporate several files as part of its source text; this is the REQUIRE declaration. Any particular separately-compiled module of the compiler begins with a series of REQUIRE declarations, each of which names a file containing some set of related definitions. The very first such file is always SYSPRM.REQ, which contains the definitions which hide the differences between the PDP-10/BLISS-10 and the PDP-11/BLISS-11 systems. The 10 version of this file is B10PRM.REQ; the 11 version is B11PRM.REQ. The first step in the changeover from one version to the other, say from the 10 version to the 11 version, is to make sure that SYSPRM.REQ is a copy of the proper version of this file (in this case B11PRM.REQ). This requires a single command to the file system.

Not all of the differences between language/machine systems could be handled by a single file of definitions, however. Those that could not may be characterized as differences between libraries. Neither of the two BLISS compilers generates code to support unsigned division, conversion of strings of characters to fixed- or floating-point numbers (or to BITS values, for ALGOL 68), or the SIGNAL/ENABLE feature of BLISS-11. These must be handled by library subroutines, and the libraries, even those which can be written in BLISS, are of a very low-level, machine-dependent nature. The two libraries are kept as separate groups of source files, and the differences in the interfaces to them are hidden in SYSPRM.REQ.

A few words of explanation of BLISS are in order before we describe the contents of SYSPRM.REQ. The data structures commonly available in other high-level languages, such as records, stacks (except for the control stack), arrays, list cells, and so forth, are not built into BLISS. To use such a structure, a program must define it, by defining
- the layout in memory of each instance of the structure. This includes a list of the parameters (e.g. dimensions) of any declaration of a variable with that structure, and specification of how those parameters are used (if at all) in determining the amount of storage allocated to that variable.

- the method by which the fields of the structure are accessed. A particular field need not be an entire machine word; then it must be specified which group of bits within the word it must be.

For instance, the standard definition of a one-dimensional zero-based array of one-word values (in BLISS-10; it is slightly different in BLISS-11) is as follows:

```plaintext
structure Vector [Index] = [Index] (.Vector + .Index) <0, 36>
```

The first pair of square brackets indicates that both declarations and accesses of Vector variables have one parameter. The expression in the second pair of square brackets indicates how large an array is allocated to such a variable when it is declared; in this case, the number of words allocated is equal to the parameter ("Index") used in the declaration. (For instance, a declaration like

```plaintext
local Vector QQQQ [445];
```

causes the array QQQQ to have length 445 words.) The remaining expression indicates how this structure is to be accessed: the location of the base variable and the value of the index are summed to get an address, and the 36 bits starting at bit 0 of the word at that address (that is, the whole word) are referred to.

The Vector structure happens to be predefined, but all other structures must be defined by the user’s program. The definition of this structure in BLISS-11 is slightly different from its definition in BLISS-10, because of the different data formats of the two machines: one can declare vectors of bytes or vectors of full words, and thus take advantage of either the byte addressing or the word addressing of the PDP-11.

This background helps to explain the organization of SYSPRM.REQ. The definitions in this file which hide differences between the PDP-10 and the PDP-11 can be grouped into four categories:

1. the definitions of "standard structures". These are:

   BYTVECT - array of 8-bit bytes  
   CHARVECT - array of ASCII (7-bit) characters  
   HFVECT - array of half words  
   HFTABLE - array of half words, from which subfields may be accessed

2. the definitions of machine characteristics. These are occasionally useful even outside the definitions of the standard structures. They are:

<table>
<thead>
<tr>
<th>QADRINC</th>
<th>PDP-10</th>
<th>PDP-11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>address increment from one full word to the next</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. the definition of the packed-string data type. This is a set of definitions, primarily of operators, for manipulating strings of characters. What makes these definitions interesting—they are the only section of SYSPRM.REQ which underwent a major design iteration—is the crucial difference between the basic string operations "read (write) a character and step to the next character", on the two machines. On the PDP-10, these operations (the IDPB and ILDB instructions) move to the next character before reading or writing; on the PDP-11, the corresponding operations (the autoincrement addressing mode) cause reading or writing before moving to the next character. This difference cannot be hidden without imposing some restrictions on the operations which may be performed on strings. We therefore divide string pointers into two categories: those for which every access of a character is accompanied by a step from one character to the next ("I-pointers") and those for which stepping to the next character must always be done separately ("N-pointers").

4. the definitions of two "library" routines which happen to be coded as macros, rather than as closed subroutines. These are the routines to clear out a block of core and to copy one block of core into another.

2. Language Differences

Neither BLISS-11 nor BLISS-10 is a proper superset of the other. Generally, however, we did not make much use of features of one language which were not present in the other, and the list of such features for which we put fake definitions in the versions of SYSPRM.REQ is uninteresting. However, three exceptions should be noted:

- The BLISS-11 SIGNAL/ENABLE construct caught our fancy, and we used it for compilation error handling. Therefore, we were obliged to write BLISS-10 constructs to simulate it. We were surprised to find that this was possible. The BLISS-11 library file, SIGEN.B.MAC, which is a standard part of the BLISS-11 compiler distribution, corresponds to the BLISS-10 library file, SYSDEP.MAC.
(which contains some miscellaneous material not related to SIGNAL/ENABLE as well). The only problem with our simulation of SIGNAL/ENABLE is that some extra code is required on normal termination of a block containing an ENABLE declaration, to pop a frame off the (conceptual) "ENABLE stack". The BLISS-11 compiler outputs this code automatically, but of course the BLISS-10 compiler does not; we put the code in the macro ENABEND, and we enforce the convention that every path to exit of an ENABLEd block must end with an ENABEND, to get around this problem.

- We wished to set up extensive preloaded symbol tables with the BLISS PLIT facility. We stumbled over one characteristic of those symbol tables, namely, that they contained many pointers to themselves and each other. The BLISS-10 NAMES and GLOBALLY NAMES feature allowed us to generate the necessary pointers easily and cleanly, but there is no corresponding feature in BLISS-11. Therefore we wrote a set of iterated and recursive macros which simulated these, as well as the related INDEXES and GLOBALLY INDEXES feature. The simulation is not quite complete, because the iterated/recursive macro processor attempts to update an index into a PLIT that it is building by counting the arguments which fly by it; it doesn't understand other features of the PLIT facility, such as duplication factors and strings. However, we did not need to make use of these features in our symbol tables.

- The BLISS-10 and BLISS-11 macro processors do not expand macro arguments in quite the same way. From our point of view, the problem is this: Let there be a macro A, with a formal parameter FA, and a macro B, with possibly some formal parameters. Suppose there is a call such as

A ( B ( ... ) )

and the call of B results in a string containing a comma—that is, a comma which is not hidden by being between a matched pair of parentheses or brackets of some sort. The BLISS-11 macro processor completes the expansion of the call of B while setting up the call of A, and since it regards the comma as an argument separator, it concludes that A has (wrongly) been passed two actual parameters, instead of just one. The BLISS-10 macro processor does not do this, treating the entire call of B as a single parameter to A.

In the ordinary course of programming, we did not encounter such problems, because of our generally conservative use of macros. However, the slightly bizarre tricks we used in the set of macros which set up the table of productions tripped across exactly this problem. We solved it by "quoting" the comma which was generated, in the BLISS-11 version, so that the BLISS-11 macro processor would not recognize it as an argument separator; this is the origin of the macro "quoted" which is defined in SYSPRM.REQ.
3. Operating System Differences

Because more than 20 complete operating systems are in use on PDP-11's and configurations of PDP-11 processors, the problem of isolating the operating system interface (or Operating System Environment, OSE) would have to have been faced even if we had not had to write a PDP-10 version of the compiler. It almost goes without saying that this problem cannot be solved by a single "REQUIRE file" of useful definitions. The TOPS-10 OSE consists of 2K words of code distributed over 2 source files (and some files from a Bliss-10 library), and the HYDRA OSE is about half as large. Thus the OSE constitutes a separate subdivision of the compiler, and, as with other subdivisions, the interface between the OSE and the rest of the compiler has been redefined several times.

Originally we thought of the OSE as a collection of utility subroutines; the compiler at the top level was a system-independent controller consisting, like many compilers and other file processing programs under TOPS-10, of an infinite loop:

```
DO
  <read input/output file specifications>
  <create object file from source file>
OD
```

Eventually we realized that the top level of the compiler must itself be part of the OSE. The system-independent portion of the compiler is, at its top level, a subroutine which compiles a single Algol source program. This seems to be the largest unit of computation which all possible OSE's can deal with, ranging from the simplest paper-tape load-and-go systems for bare PDP-11's to complete operating subsystems with their own text editors, linkers, and version backup systems. Thus the HYDRA OSE consists of:

1. A set of utility subroutines, invoked from various points in the system-independent part of the compiler. These are:

   1a) OSEsrcchar
   Reads a single character from the source file, and returns the character as its value. End-of-file is denoted by a special character. Note that end-of-file here means end-of-logical-source-program; for instance, in the TOPS-10 OSE (and perhaps in future HYDRA OSE's) the source program compiled in a single compilation may stretch over several of the entities that the operating system knows of as "files".

   1b) OSElstline
   Sends a single line of text off to be listed. The system-independent portion of the compiler believes that there are two places to which text output may be sent: a "listing device" and a "command device", corresponding in many systems to a line printer and a user's terminal, respectively. Thus one of the arguments to this routine is an indicator of which place(s) the line is to be sent to.

   1c) OSEerrmsg
   Outputs a single error message. The sole argument to this
(1d) **OSEobjword** Outputs a single word of code to the object file. This would be too small a unit of output if the compiler had to deal with different loader formats for such things as fixups, relocatable segments, and overlay structures, but we have avoided those by writing our own loader. **OSEobjword** is a null routine in the TOPS-10 version.

(2) A “top level”. This opens various standard channels for I/O, performs other system-dependent initializations (e.g., initializing the symbolic debugging system), and calls the system-independent compiler. In load-and-go systems, this top level might start the user’s program when the system-independent compiler had finished. (Alternatively, this might be more conveniently left to the operating system itself.) The current HYDRA OSE top level also includes:

(2a) A routine to handle aborted compilations. The system-independent compiler may find itself unable to continue due to such things as internal errors, lack of core, or I/O failures; in such cases it returns to its caller, with a result value indicating the cause of failure.

(2b) Routines to initialize the settings of various compilation options, based on information supplied by the user to the operating system and passed by the operating system to the OSE.

(2c) A routine to clear out a workspace in core for the compiler; this initialization must be in the OSE, because different versions may have completely different arrangements in memory of the compiler’s code, workspace, and internal control stack.

(3) A set of global variables, which serve as an additional means of communication between the compiler and the OSE. They are:

(3a) **GBLerrs** Count of ordinary compilation errors.

(3b) **GBLwarns** Count of “mild” compilation errors—those which do not prevent the compiler from generating code.

(3c) **GBLfreeo** Points to the start of the compiler’s workspace.

(3d) **GBLfreeh** Points to the end of the compiler’s workspace.

(3e) **GBLprogs** An indication of the starting address of the compiled program.

(3f) **GBLpragflags** The area (currently a single word) in which all the compilation option settings are kept.
4. Conclusions

There is a natural conflict between, on the one hand, the universal language design goal of portability and, on the other hand, one of the goals of system implementation languages (SIL's) in particular, namely that the programmer should be able to make use of all the facilities of the hardware, such as unusual instructions. The designers of PL/360[6] have resolved this conflict by sacrificing portability altogether. The designers of MARY[7] describe a subset of the language, "safe MARY", and a set of features which extend it to "unsafe MARY"; programs written entirely in the subset are guaranteed to be portable from one computer to another. The compiler can enforce the restriction to the subset, if the programmer desires. The approach taken by BLISS is not quite so strict as this. The designers regarded the various dialects of BLISS as "a class of languages that are similar in philosophy and that mirror a similar concern for the important aspects of systems programming, but each of which is tailored to its host machine[3]."

Usually the requirements of portability cannot be ignored. The more successful SIL's are bound to be implemented on more than one kind of computer, and for all but the lowest-level, most hardware-dependent programs (device handlers, diagnostics, and so forth) it will always be attractive to copy a program already written, even from another computer, rather than to write a new program from scratch. Therefore it is useful to consider what aspects of BLISS we found helpful in constructing a portable program, and what aspects were useless or even harmful.

It is evident that our principal tools, namely the REQUIRE feature and the practice of keeping the two libraries in separate groups of files, are extremely powerful; they could be used, and were used, as a "last resort" in the solution of portability problems, when there was no way to avoid writing two different (machine-dependent) versions of some routine or group of routines. The effectiveness of our techniques is therefore judged by the extent to which we had to use the "last resort", i.e. by the size of the libraries. Indirectly we also judge the suitability of BLISS for writing portable programs by this criterion.

In this respect the STRUCTURE definition facility has been outstandingly successful. The string data-type is an example of what has been done with this. Strings are represented in completely different ways on the two machines: packed 5 characters to a word on the PDP-10, aligned on byte (half word) boundaries on the PDP-11. But by the use of the CHARVECT structure, together with a small set of operations based on the BLISS-10 "special functions" SCANN, REPLACE!, etc., we have made string manipulation completely uniform, so that no library routines whatever had to be written for it. Perhaps an even more striking example is that of the structures PBLOCK and OBLOCK, the standard structures onto which pointers to blocks of packed data are mapped. These did not even have to be defined in SYSPRM.REQ, and thanks to the macros RH, LH, WORDF, ADDR, and SPCF, the definitions of fields in the various types of blocks need not be placed in the machine-dependent REQUIRE file either.

The LINKAGE definition facility should also be mentioned although we have so far made little use of it. By defining a linkage, such as REGO, one imposes special calling
conventions (e.g. special locations for actual parameters) on all routines which are declared with that linkage. It is particularly well designed for portability, in the following respects:

- Code which makes use of LINKAGE definitions and is correct is guaranteed to be correct when the use of special linkages is removed.

- The use of special linkages is particularly easy to remove. That is, the "fake definitions" which one must put in one version, corresponding to the linkage definitions and uses of another version, are extremely simple. The definition of REGO in the BLISS-10 version, our only example, is a null macro.

Two features of BLISS which we had to essentially ignore (except in the construction of libraries) were the ability to descend into machine code (the MACHOP feature of BLISS-10, the OPCODE feature of BLISS-11) and the ability to force local variables to be allocated to the machine's fast registers (the REGISTER declaration). It is worth asking why we could not make use of the second of these.

The alternative to the REGISTER declaration, in both versions of BLISS, is the LOCAL declaration. The choice between these two alternatives is not quite the same in BLISS-10 as it is in BLISS-11. In the former, LOCAL variables are always allocated on the control stack, but in the code that is generated, copies of the variables are often put in registers to allow easier access to them. In the latter, any LOCAL variable may be allocated either to a register or to a location on the stack, depending on the whim of the compiler, which attempts to find an optimal allocation. In both cases, the usefulness of declaring a variable as REGISTER instead of LOCAL in order to make frequent accesses to it less expensive is clouded or even nullified by the actions of the compiler. Thus REGISTER declarations are primarily useful only when they are necessary, e.g. in conjunction with use of the MACHOP (OPCODE) feature.

This paper has not covered two aspects of the question of portability which may be of general interest. The first of these is portability between really different languages. Although a version of the compiler has been produced principally by transcribing the BLISS source code into another (unrelated) higher-level language (PASCAL), we were not sufficiently familiar with that effort to discuss the questions raised by it in this paper.

The second of these is portability of the compiler to installations where BLISS compilers are not available. Since the BLISS-11 compiler produces assemblable PDP-11 code, it is possible to export the PDP-11 version of the compiler in the form of a set of assemblable files. But this leaves little room for the user of the system to make local modifications to it, such as additions of new language features, or modifications to the compiler's pre-initialized symbol tables; it is difficult or impossible in most cases to follow the assembly code produced by the BLISS-11 compiler, since it is, after all, completely uncommented and unformatted, and labels in the assembly code do not correspond in general to labels in the BLISS-11 source code. We have not yet attacked the problems raised by the general unavailability of BLISS-11 compilers at user sites, and so we postpone the discussion of them to future articles.
References


Appendix

The following files are included here in their entirety:

**B10PRM.REQ** - PDP10 version of SYSPRM.REQ
**B11PRM.REQ** - PDP11 version of SYSPRM.REQ
**COMMON.REQ** - REQUIRE file used by every module in the compiler

**SYSDEP.MAC** - support routines for BLISS-10 SIGNAL/ENABLE
**SIGENB.MAC** - support routines for BLISS-11 SIGNAL/ENABLE

**A68S10.B10** - heart of the TOPS-10 OSE
**A68S11.B11** - heart of the HYDRA OSE

**MDSTRC.REQ** - typical REQUIRE file
**LEXAN .BLI** - representative code module
**SRTABS.BLI** - representative use of the DATAAREA macros
Copyright 1977 P. Hibbard and P. Knusven
Pittsburgh, Pennsylvania

System-dependent declarations for the Bliss II version.

bind
quadinc = 1,  // Increment from word address to next word address
quadsize = 16, // Number of bits in an address value (pointer)
quadword = 5,  // Number of characters per word
quadsize = 16, // Number of bits per half-word
quadtab = 1,   // True if horizontal tabs may be sent to output devicep
gapsed = 35,   // Must be 0 or Quadsize-1
quadsize = 36, // Number of bits per word

Macros to supply actual names of the Operating System Environment
routines and globals.

macro
GBL errs = ?.errs $;
GBLwarns = ?.warns $;
GBLprogflags = ?.progflag$;
GBLprogtart = ?.progstart$;
GBLfreshi = ?.fresh $;
GBLfreeo = ?.freeo $;
OSEsrcchar = ?.SRCchar $;
OSElineline = ?.LINELINE $;
OSEobjword = ?.OBJword $;
OSEerrmsg = ?.Errmsg $;

Macros to smooth over the differences in the Bliss-10 syntax and
the Bliss-11 syntax.

macro signal(a)=
  begin external ?.SIGNL;
  (?Freg-1)<8,36> = ?.SIGNL<8,0>;
  return a
end $;

enable=
  register R(5);  // Forces all declarable regs to be preserved.
  local ?.Frame(4);
  external ?.ENABL,Signal, Sigreg;
  Vreg = (?Freg-1)<8,0>;
  ?.ENABL();
  if .Vreg eq 0
    then 0
    else exit block select .Signal of
         next $,
end enable;

elsename=
Otherwise: signal(.Signal)

ten $,

enaband

Sigreg-8(.Sigreg-2) $

enableave

| The programmer is cautioned against

enabband; leave $,

| using enableave or enabreturn to

enabreturn

| exit more than one level of ENABLEd

enabband; return $,

| blocks; an extra enaband is required

| for each level.

byte = $,

quoted = $,

uplit = plit $,

unname = list $,

stacklocal = local $,

Reg0 = $,

maxi(A,B) = if (A) geq (B) then (A) else (B) $,

mini(A,B) = if (A) leq (B) then (A) else (B) $,

issu = iss $, lequ = leq $,

equ = eql $, nequ = neq $,

gequ = geq $, gtru = gtr $,

rebindli(N,V) =

bind XXXtmp = V;

undeclare N;

bind N = XXXtmp;

undeclare XXXtmp $,


/ Handy macro in Bliss-10

/ ID(x) = x$,

/ Create pointer which consists of the address OR'd with the
/ operand bit. This cannot be done in a straightforward way
/ due to a Bliss-10 bug.

/ SpecPtr(x) = (x)<32,8> $,

/ Cheap non-zero value

/ nonzero = .Sreg $,

/ Some attempts at solving the problem of declaring data in a
/ machine-independent, language-independent way.
/ Major obstacles to be surmounted:
/ GLOBAL BINDs to symbols are illegal in Bliss-10;
/ Bliss-10 does not have initialized ONVs or GLOBALs;
/ Bliss-11 does not have NAMES, INDEXES;
/ Bliss-10 does not have iterated or recursive macros.

/ globaldata(strc,nm,sz) = bind strc ID(QQQ)nma(sz)=plit(n) globally names $,
/ dataarray(n) = bind Vector ID(QQQ)n = plit(n) globally names $,
/ dataend = $ $,
/ gblname(name,d) = name globally names d $,
gbindex(name,d) = name globally indexes d $,
locname(name,d) = name names d $,
lcndex(name,d) = name indexes d $,
data(d) = d $,

gbname(x) = x globally names $,
gibbind(x,v) = switches optimize $,

| Support for machine-independent scan and replace operations |
| See BIOPRM.REQ for more explanation. |

Hptrtochars(x) = (x)<29,7> $,
Hptrtobytes(x) = (x)<28,8> $,
Iptrtobytes(x) = (x)<26,7> $,
Iptrtochars(x) = (x)<26,8> $,
ItoIntr(x) = (x) $,
Iptrntr(x) = (x) $,

Pl Perform zero and move operations in best machine-dependent way.

Zerocor(start,cnt)=
begin machop BLT=#251;
register R1
R={start+1}; R<18,18>({start});
start=0;
BLT(R,(start)+((cnt)-1));
novalue end $;

Movecor(s,d,cnt)=
begin machop BLT=#251;
register R1
R={d}; R<18,18>({d});
BLT(R,(d)+((cnt)-1));
novalue end $;

| Some structures. |
| structure Bytevec[1] = ((1+3)/4) (.Bytevec+.1/4)<28-8e(.1 and 3),8> |
structure Chrevct[1] = ((1+4)/5) (.Chrevct+.1/5)<29-7f(.1 mod 5),7> |
structure Hfvec[1] = ((1+1)/2) (11 .1 then (.Hfvec+.1/2)<18,18> |
| else (.Hfvec+.1/2)<0,18>) |
structure Hftable[1, J, P, S] = ((1+1)/2) (11 .J then (.Hftable+.1, J/2)<18>,P, S> |
| else (.Hftable+.1, J/2)<0, P, S>) |

| END OF BIOPRM.REQ |
Copyright 1977
P. Hibbard and P. Knueven
Pittsburgh, Pennsylvania

System-dependent declarations for the Bliss II version.

bind
- Qadrcn = 2;  // Increment from word address to next word address
- Qadrsiz = 8;  // Number of bits in an address value (pointer)
- Ochowrd = 2;  // Number of characters per word
- Ohtsz = 6;    // Number of bits per half-word
- Ohtab = 0;    // True if horizontal tabs may be sent to output device
- Osparo = 0;   // Must be 0 or Oqyksiz-1
- Oqyksiz = 16; // Number of bits per word

Macros to supply actual names of the Operating System Environment
routines and globals.

macro
- GBlorrr = $Errs $;
- GBlwrrs = $Wrrns $;
- GBlfreelo = $Frello $;
- GBlfreeshi = $Freeshi $;
- GBlprostart = $Prostart $;
- GBlproflags = $Proflags $;
- OSEcrcher = $SCRcher $;
- OSElistline = $SLSTline $;
- OSEobjword = $OBJword $;
- OSEerrmsg = $ERRmsg $;

Macros to smooth over the differences in the Bliss-10 syntax and
the Bliss-II syntax.

macro
- enabend = $;
- enableave = leave $;
- enabreturn = return $;
- quoted = $Quote $Quote $;
- maxi(R,B) = ((R) max (B)) $;
- mini(R,B) = ((R) min (B)) $;
- semantic = $;
- rebindt(N,V) = rebind N = V $;

Create pointer consisting of address OR'd with the Osparo bit.

SpecPtr(x) = (x) OR $;

Chomp non-zero value.

nonzed = .PC $,
Sets attempts at solving the problem of declaring data in a
machine-independent, language-independent way.

Major obstacles to be surmounted:
- GLOBAL BINDS to symbols are illegal in Bliss-10;
- Bliss-10 does not have initialized OSVs or GLOBALS;
- Bliss-10 does not have NAMES, INDEXES;
- Bliss-10 does not have iterated or recursive macros.

Macros whose names begin with $ (S$atdata, S$setupdata, S$strippars)
S$setupbinds) are not intended to be used outside BILPRM.REQ.

```
globaldata(src, nm, sz) = global bind src nm(sz) = uplit( src, dataarea(name) = S$atdata(name, S, 
dataend = ) S, 
gblname(name)[i] = 
Squote2 S$namebind, global, name, S$length-1, (S$remaining) S, 
gbindex(name)[i] = 
Squote2 Sindxbind, global, name, S$length-1, (S$remaining) S, 
lcname(name)[i] = 
Squote2 S$namebind, name, S$length-1, (S$remaining) S, 
lcindx(name)[i] = 
Squote2 Sindxbind, name, S$length-1, (S$remaining) S, 
data[i] = 
Squote2 S$nobind, 
S$atdata(name)[] =
    external name;
    bind S$name('.$NAME.') = name, S$name('.$INDEX.') = 0;
S$setupbinds(S$remaining);  
undecare name, S$name('.$NAME.'), S$name('.$INDEX.');
global bind name = uplit S$setupdata(S$remaining) S, 

Squote2 = quoted quoted quoted S, 
S$namebind(name) = bind name = S$name('.$NAME.') + 2aS$name('.$INDEX.') S, 
Sindxbind(name) = bind name = S$name('.$INDEX.') S, 
S$nobind(name) = switches optimize S, 
S$setupdata(bnd, gbl, name, len, list) = S$strippars list S, 
S$strippars() = S$remaining S, 
S$setupbinds(bnd, gbl, name, len, list) = 
    gbl bnd(name); 
    rebind S$name('.$INDEX.') = S$name('.$INDEX.') + len $, 

gblname(x) = S, 
gbibind(x, v) = global bind x = v S, 
```

Support for machine-independent scan and replace operations

The goal is to have efficient character/byte scanning operations
which are machine independent. Bliss-10 provides scan and replace
operators which allow the programmer to perform POP-10 byte instructions
and thus are relatively efficient. An efficient method of scanning on
the PDP-11 is through the use of auto-increment addressing modes in
byte instructions. Thus, an obvious approach to reaching the goal is
to provide Bliss-11 scan and replace macros which cause the generation of
byte instructions with auto-increment operands. The major difficulty
in implementing this solution is that the POP-10 pointer-incrementing
Instruct ions perform the increment before a fetch or store, while the PDP-11 instructions increment afterwards. This difficulty is overcome by forcing the programmer to pay special attention to the way in which pointers are initialized and used.

Pointers may be one of two varieties, I-pointer or N-pointer.
N-pointers always refer to a particular byte until an incp is performed.
I-pointers increment before each access, i.e. after an access an I-pointer refers to the byte accessed. The rules for using these kinds of pointers may be stated by consistently substituting N or I for X in the following paragraph.

X-pointers must be initialized by assigning a value yielded by an Xptrto... operator. All operations on an X-pointer must be an incp, X-operation (i.e. scanX, replaceX or copyX) or the appropriate conversion operator.

An I-pointer value may be converted to an N-pointer value by using ltoNptr. The new N-pointer refers to the byte last accessed by the I-pointer. Similarly, NtoIptr may be used to create an I-pointer from an N-pointer. The new I-pointer refers to the same byte as the N-pointer. However, since only I-operations are permissible that byte is not accessible via the I-pointer.

lptrndx(n) is used in conjunction with the Bytovect and Charvect structures. When used as the actual parameter in a Bytovect or Charvect structure access, it produces an I-pointer which will access the n-th byte or character in the first I-operation in which it is used.

Perform zero and move operations in best machine-dependent way.

ZeroCor(adr,cnt)=
begin
register Ptr;
opcode CLR;
Ptr=(adr)+c(cnt)*2;
while .Ptr gt ru (adr) do
  CLR(8-Ptr);
no value
end $;

MoveCor(a,d,cnt)=

begin register Src, Dst;
opcode MOV;
Src:=1; Dst:=1;
decr I from (cnt)-1 to 0 do
    MOV(Src, Dst);
end

Some structures.

structure
    Byt vect[1] = [1] (.Byt vect[1]<>0,1>,
    Char vect[1] = [1] (.Char vect[1]<>0,1>,
    Hf vect[1] = [1] (.Hf vect[1]<>0,1>,

Some linkages.

linkage
    Reg0 = Bliss(register=0);

END OF 81PRMH.REQ
Common definitions included in every module.

Target machine parameters

bind

\begin{align*}
X_{base} & = 0, \\
Y_{base} & = 100000, \\
Z_{adrinc} & = 2, & \text{Increment from word address to next word address} \\
Z_{chars} & = 2, & \text{Characters per word} \\
Z_{hsize} & = 8, & \text{Bits per half-word} \\
Z_{pare} & = 1, & \text{Value to add to make an illegal word address} \\
Z_{rdsize} & = 16, & \text{Bits per word} \\
Z_{rsizNat} & = 2, & \text{Size to add to make an illegal word address} \\
Z_{rsizLhbase} & = 7, \\
Z_{rsizLhbase} & = 18, \\
Z_{rsizLhtop} & = 40 \text{if } Z_{rsizNat} \text{ eqi } 1 \text{ then } 2, \\
Z_{rsizDressed} & = 2, \\
Z_{typConblock} & = 74, \\
Z_{rsizDesctrip} & = 41.
\end{align*}

Useful keywords

\text{macro}

\begin{align*}
thn & \text{ then if } S, \\
elseif & \text{ else if } S, \\
else & \text{ always } S, & \text{To be used like "T" in a Lisp COND.} \\
repeat & \text{ while } L \text{ do } S, \\
exitL & \text{ leave } L \text{ with } S, \\
numer & \text{.Vreg } S, \\
bool(x) & \text{ if } (x) \text{ then true } S, \\
times3(x) & \text{ inc } S, \\
\text{modulo}(L,R) & = (1 \text{ if } ((R) \text{ and } -(R)) \text{ eqi } (R) \\
& \text{ then } (L) \text{ and } (R)-1 \
& \text{ else } (L) \text{ mod } (R) \text{ ) } S, \\
\text{divide}(L,R) & = (\text{if } (R) \text{ eqi } 1 \text{ then } L) \\
& \text{ elif } (R) \text{ eqi } 2 \text{ then } (L)/(R)-1) \\
& \text{ elif } (R) \text{ eqi } 4 \text{ then } (L)/(R)-2) \\
& \text{ else } (L)/(R) \text{ ) } S, \\
\text{Npstrchars} & \text{(split ascii str) } S, \\
\text{comment} & \text{ switches optimize } S.
\end{align*}

Some structures

\begin{align*}
\text{structure Bitvect(P,S)} & = (.Bitvect <.P,.S>), \\
\text{structure Pblock(I,P,S)} & = (.Pblock <Xadrinc,I> <.P,.S>), \\
\text{structure Oblock(I,P,S)} & = (.Oblock <Xadrinc(I-1)> <.P,.S>), \\
\text{structure Constbv(I,J,K)} & = (\text{if } .K \text{ eqi } 0
\end{align*}
I More useful definitions

bind
  true = 1,
  false = 0,
  zed = 0,
Constbv Minus = -1,
  HTab = #11,
  LF = #12,
  FF = #14,
  CR = #15,
  EOFchar = #208,
SPLStacksize = 128, /* Semantic/Syntax stack size
Cbufsize = 132; /* Standard character buffer size

I Macros useful in field definitions

macro
  ZRH = 0, Zhtsz $,
  ZLH = Zhtsz, Zhtsz $,
  RH = 0, Ohtsz $,
  LH = Ohtsz, Ohtsz $,
  Wordf = 0, Owordsz $,
  Addr = 0, Quadsz $,
  Spec = 0, Spare, 1 $,
  Cmpspare(x) = (x) and not 1stSpare $,
  Zhtwords(Lo, Hi) = ((Hi) * Zhtsz or (Lo)) $,
  Ntwords(Lo, Hi) = ((Hi) * Ntysz or (Lo)) $;

I Fatal error codes

bind
  FatNospace = -1,
  FatBufovfl = -2,
  FatUser = -3;

I Interface to Operating System Environment (OSE)

external
  GBLerrs, GBLwarns, 
  GBLfreelo, GBLfreeshi, 
  GBLprostart, 
  Bitvect GBPlprflags, 
  OSEarchar, 
  Reg0 OSElistline, 
  Reg0 OSEobjword, 
  Reg0 OSEarrmsg;

I Field definitions for GBPlprflags

macro
  Prgstrop = RH $,
  Prglst = Ohtsz, 0, 1 $,
<table>
<thead>
<tr>
<th>Pryobj</th>
<th>Ohtsz=1,1 $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prgwarn</td>
<td>Ohtsz=2,1 $</td>
</tr>
<tr>
<td>Prgwach</td>
<td>Ohtsz=3,1 $</td>
</tr>
<tr>
<td>Prgnonstd</td>
<td>Ohtsz=4,1 $</td>
</tr>
<tr>
<td>Prgnghost</td>
<td>Ohtsz=5,1 $</td>
</tr>
</tbody>
</table>

END OF COMMON.REQ
TITLE SYSDP ALGOL60 SYSTEM DEPENDENT MACHINE LANGUAGE ROUTINES
THOSEG

JOBREN=124
LOC JOBREN
XWD 0,RENADR

JOB41=41
LOC JOB41
IFCL ; NOP FOR SIX12 UD0'S

RELOC 400000

COMPILER CONSTANTS
LSTCHN=1

CALLI DEFINITIONS
RESET=8
EXIT=12

CODE EXECUTED WHEN MONITOR "REENTER" COMMAND IS GIVEN
RENADR: CLOSE LSTCHN,0 ; CLOSE LISTING FILE ON REENTRY
CALLI 0,RESET ; DO NOT WRITE OTHER FILES
CALLI 0,EXIT ; GO AWAY

BLISS 10 SPECIAL REGISTERS
SS=0 ; SREG: STACK POINTER REGISTER
SF=2 ; FREG: POINTS TO CURRENT INVOCATION BLOCK
SV=3 ; VREG: VALUE REGISTER

DETECT FLOATING-POINT OVERFLOW
FLTOVF:
SETO SV,
JFOV 2
SETZ SV,
P0PJ SS,
ROUTINE TO SET UP ENABLE FRAME

INPUTS
- VREG - Pointer to new 4-word frame

OUTPUTS
- SIGREC - Pointer to frame 3
- VREG - Zero

.END:
  EXCH $5,1
  PUSH $V,0(1)
  PUSH $V,SIGREC
  PUSH $V,$F
  PUSH $V,1
  S0S 0($V)
  MOVEM $V,SIGREC
  SETZ $V,
  EXCH $5,1
  POPJ $5,

ROUTINE TO PERFORM A SIGNAL

INPUTS
- VREG - Signalled value

OUTPUTS
- SIGNAL - Signalled value
- SIGREC - Restored to point to previous frame
- VREG - Non-zero
- Returns to most recent (dynamically) enable declaration

.SIGNAL:
  MOVEM $V,SIGNAL
  MOVE $V,SIGREC
  POP $V,$S
  POP $V,$F
  POP $V,SIGREC
  POPJ $V,

; INITIAL ENABLE CODE
ENABL: CALLI 1,EXIT

; INITIAL ENABLE FRAME
FRAMEB: EXP ENABL,FRAMEB+3,0,0

RELOC 0

; VARIABLES USED BY SIGNAL/ENABLE Routines
SIGNAL: 0 ; Signalled value
SIGREC: FRAMEB+3 ; Points to top of stack of enable frames

END
The BLISS-11 out of line routines to handle SIGNAL's and ENABLE frame creation.

P. Knuenen
26-Jul-76
P. Kariton - change to location of SIGREG and SIGNAL to be $NVREG and $NVVAL as in STKPRG.REG(1H81HY97)
Commented out LEVO and EXIT also.

The dynamically nested occurrences of ENABLE declarations are recorded in the LIFO ENABLE stack. This is implemented as a linked list of 3-word stack frames. The current top of the stack is pointed to by .SIGREG. An ENABLE frame is created each time an ENABLE declaration is "executed".

It looks like the following:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pointer to most recent previous frame</td>
</tr>
<tr>
<td>2</td>
<td>Value of SP for the ENABLE body</td>
</tr>
<tr>
<td>4</td>
<td>Pointer to the ENABLE body</td>
</tr>
</tbody>
</table>

!*GLOOL $SIGNL,$SIGN1
GLOOL $ENABLE
GLOOL $VALUE,$SIGREG

Calling Sequence
MOV E,R0
JMP $SIGNL

$SIGNL: MOV R0,$VALUE
$SIGN1: MOV $SIGREG,R0
MOV (R0)+,$SIGREG
MOV (R0)+,SP
MOV (R0)+,PC

Calling Sequence
MOV $=6,R0
JSR PC,$ENABLE
.WORD L2-L1
L1:
... ENABLE body ...
where \( n \) is 2 plus the offset into the stack of the space reserved for the frame

```
SENAOL: ADD SP, R0
        MOV (SP), (R0)
        ADD #2, (R0)
        MOV @SP, (SP)
        ADD (R0), (SP)
        MOV SP, -(R0)
        ADD #2, (R0)
        MOV SIGREG, -(R0)
        MOV R0, SIGREG
        RTS PC
```

;SEXIT: HALT
;
;SLEV0: .WORD 0,0,SEXIT

SIGREG = 270 ; SAVREG address
SIGVAL = 272 ; SAVVAL address

.END
module AGS10(stack(1000)) = begin

    POP-11 ALGOL 68S Cross-compiler for POP-11

    switches null;
    require DIOPRM.REQ;
    require COMMON.REQ;
    require IOMACS.REQ;
    switches list;

    undeclare
        OSElistline,
        OSEerrmsg,
        OSEobjword,
        OSEarchar;

    forward
        Initchnl,
        InitFOB,
        Endmsg,
        Cvtfile,
        Reset,
        Getbuf,
        Lkupnr,
        Getcmd,
        Parseg,
        Initchna,
        Punt,
        Page,
        Lread;

    bind
        FatCmdLong = FatUser-0,
        LSTlinesperpage = 56;

    own
        LSTheadpg,
        LSTpage,
        LSTcount;

    Machop CallI = #47;

    bind
        Crlm = Mag("M7J"),
        Spscm = Mag("'"),
        Errm = Mag("Command error"),
        PPNm = Mag("Invalid PPN");
External

Buffers,
Close,
Date,
Dayofweek,
Enter,
Filename,
Lookup,
Open,
Pdate,
Prime,
Purgeout,
Read,
Write;msg;

Own

10buff1(10buff),
10bufp,
Srcbufp,

Bitvec Create,
FD0(5),
Extensible,
Chnl;

Bind

Device = F0D(0),
Filename = F0D(1),
Extension = F0D(2),
Pr-TiDate = F0D(3),
PPN = F0D(4),

LlupBlk = F0D(1),
NtrBlk = F0D(1);
structure Bufhdvoc{I} = { [3e1] (.Bufhdvoc+3e.1) }
structure Block{I,P,S} = { IsQadrinc1 (.Block+Qadrinc.1)<P,.S> }

own Bufhdvoc BufhdvocHichn1; 
own Block BlockHichn1;

structure Chnstatwd{P,S} = { Chnstate+Chnstatwd<P,.S> }

map Chnstatwd Chnl; so that .Chni{Specdl} means .Chnstate{Chni,Specdl}

bind Vector Initstat{Hichn1} = pItt( 
  Hfwords(0,1), 
  Hfwords(0,2), 
  Hfwords(0,2));

macro Initchn{chan,blk} = 
  begin 
    blk is always FDB 
    Chnl{chan}; 
    Lkupntr(); 
  end$;

macro InitFDB{dev,fil,e,ext} = 
  begin 
    Device=dev; 
    Filename=fil;e; 
    Extension=ext; 
    PrTlDate=PPN-B; 
  end$;

comment Cvftiile
  
  Functions: set up a character string, to be output by the caller, 
  that describes a file (device, name, extension).
  
  Locals
    Ndbuf - space to hold the string 
    Dstp - a byte pointer to the string 
    FDB - the block holding the information about the file
  
  Output
    VALUE - byte pointer to the 1st character of the string
  
  routine Cvftiile = 
    begin 
      own Ndbuf{4},Dstp;
      
      comment Cvtaix(N)
        
        Functions: convert a SIXBIT string to ASCII and append it to 
        the current string (DSTP).
        
        Inputs
          N - word containing the SIXBIT string
routine Cvtslx(N) =
begin
local Srcp;
Srcp=N<36,S>;
dec 1 from 5 to 0 do
if scan(Srcp) eq 0
then exit loop
else replace (Dstp,scan(Srcp)+#40);
end;
Cvtslx(.Dev lc e);
Replace (Ost p,.);
Cvtslx(.F ilena eo);
Replace (Ost p,0);
Cvtslx(.Exten sion and #7777777Qhtsz)
Replace (Ost p,0);
Nptrtochar(Hasbuf)
end;

comment I Cmdmsg(IStr)
| |
| ! Function: output a message to the user's terminal, ending |
| ! with CRLF, using TTELL's. |
| |
| ! Inputs |
| ! IStr - pointer to a word whose high byte begins the |
| ! string to be output. |
| |
| rout ine Cmdmsg(IStr) =
| begin
| Writems (Chnchn,IStr);
| Writems (Chnchn,Crlf);
| Purgoout (Chnchn);
| novalue
| end;

comment I Reset
|
| ! Function: do a RESET WITH, and reinitialize the Chnlint words |
| |
| rout ine Reset =
| begin
| map Vector Chnlint;
| 10buf=10buf<8,8>;
| Novector (Initstat,Chnlint,Hichn+1)
| Calli (0,0);
| novalue
| end;

comment I Getbuf
|
| ! Function: allocate space for I/O buffers for a given channel,
calling the BLLIB routine BUFFERS.

**Inputs**
- Chnl - a channel number, between Cnchn and Hichn inclusive
- Iobuff[Iobuf] - a large block of space from which all I/O buffers are taken
- Iobufp - a pointer to the beginning of unused space in Iobuff

**Outputs**
- Iobufp - updated
- VALUE - True if buffer space was allocated, false if some error was encountered

**routine Getbuf**

```plaintext
test
begin
local Size;
if .Chnl eql Srcchn then Srcbufp .. Iobufp;
Size = 2x(2xBuffers(.Chnl, 0, Chnl[Inpio], 0));
if . Iobufp . Size leq IObuff[Iobuf] then
    begin
    Buffers(.Chnl, 2, Chnl[Inpio], 10bufp); 10bufp .. Iobufp + Size;
    True
    end;
end;
```

**comment**

**Function:** Initialize a channel for I/O as indicated

**Inputs**
- Chnl - the channel number
- FOB - a block containing all necessary information about the current channel, except that contained in Chnlstat[Hichn] - a vector of words, one for each channel, each of which contains a few bits of information about the channel.

**Outputs**
- VALUE - True if initialization succeeds, false otherwise

**routine Llupnt**

```plaintext
test
begin
macro Devchr(dvml) = (register QQ; QQ-dvml; call(QQ, #4));
macro BinaryInt = 12, 1 S;  // Names of fields in the DEVCHR result word
    AscInt = 1, 1 S;
    77Devnt = 21, 1 S;
bind Binarymode = #14, Ascimode = 1;
bind L158 = #25627, L158AL71 = #2562710165;
register Bitvect A;
if not .Chnl(Spcccl) then .Chnl(Spcccl): ! Is channel already in use?
    if .(R.Devchr(,Device)) neq 0 : ! Does device exist?
        if .(if .Chnl(,Binio) then .R(BinaryInt)
            else .R(AscInt))
            then .Open(,Chnl), ! Does OPEN UUO succeed?
```

**Inputs**
- Chnl - a channel number, between Cnchn and Hichn inclusive
- Iobuff[Iobuf] - a large block of space from which all I/O buffers are taken
- Iobufp - a pointer to the beginning of unused space in Iobuff

**Outputs**
- Iobufp - updated
- VALUE - True if buffer space was allocated, false if some error was encountered
then Binarymode
else Ascillmode,
if .Chnl[Inp] then Bufnd(.Chnl)<0,8>
else Bufnd(.Chnl)<0,8>0?0false

if Getbuf() neq 0    ! Can buffers be gotten?
if (.Chnl[Inp])    ! Does LOOKUP or ENTER UDO succeed?
then begin
  local SavPPN;
  SavPPN=PPN;
  if Lookup(.Chnl,UpBlk) then exitblock true;
  if not .Exitseen then
    begin
      Extension=0;
      if Lookup(.Chnl,LipBlk) then exitblock true;
      Extension=sixbit '88';
      end;
      if .SavPPN eq 0 and .GreatlwLH eq L150 then begin
        PPN=L150AL71;
        if Lookup(.Chnl,LipBlk) then exitblock true;
        end;
    false
    end
  else Enter(.Chnl,HtmlBlk))
  then begin    ! Success
    Chnl[TTYd]=-RTTYdevi;
    Chnl[Speccl]-True;
    return True;
    end;
Cmdmsg(Hng("Initialization failure"); ! Failure
Cmdmsg(Cvtf11.0);
False
end;

comment ! Getcad
!
! Function: input a command from the user's terminal,
! using TCALL's
!
! Outputs
! Cadbuf - a block of bytes in which the command is stored as an ASCII string
!
routine Getcad =
begin
  bind Cadbuf = 20;
  own Cadbuf[Cadbuf];
  register Ptr,W;
  Ptr=Ptrtochars(Cadbuf);
  N=Ocharenc(Cadbuf);
d0 begin
  if (N-W) leq 0 then
    Punl(FatCadlong);
    replace1(Ptr,Read(Chnl));
  end
end;
until .Vreg eq LF;
inputchars(Cadbuf)
end;

routine Parse =
begin
macro GetPPH = (register QQ: QQ=0; Calli(QQ,#24));
macro Greet(Who,What) = Who: exit; select Cadbuf(What);$;
bind
AH20 = #095040,
AL60 = #010130,
AL90 = #010131,
AL70 = #010164,
AL74 = #010170,
BL03 = #032013,
GA10 = #155604,
PH02 = #427112,
PK02 = #431202,
SA20 = #511150,
TR30 = #547404,
WM17 = #641573;
if .Greet noq 0 then return novalue;
Greet=GetPPH;
select .Greet(RH) of
next
Greet(AH20,'Watch me blaze Pascal');
Greet(AL60,'Let me elaborate on that');
Greet(AL90,'Let me elaborate on that');
Greet(AL70,'Let me elaborate on that');
Greet(AL74,'Where's the missing link??');
Greet(BL03,'Hi Bruce');
Greet(GA10,'Huy Guy');
Greet(PH02,'Thou art Peter');
Greet(PK02,'Good luck Paul');
Greet(SA20,'It's that crazy Swede again');
Greet(TR30,'Give my regards to Cas');
Greet(WM17,'It's big daddy Bill');
next;
novalue
end;

comment ! InitchnaSrc only
|
| Function
| Initialize I/O channels.
| ! If Src only is eq 0 then get command line from user and initialize specified channels.
| ! If Src only is lss 8 then initialize source input channel to read from next specified input file. Return False as value iff end-of-data or error occurred.

routine InitchnaSrc only =
begin
macro

```plaintext
Caddone = return True S,
Caddfail = return False S,
Caddorr(x) = (Caddmsg(x) ; Caddfail) S;

own Caddof,Ptr;
local Bitvec Scanv;
macro Retcode = RH S,
Breakch = LH S;

if .Srconly geq 0
then begin
Reset();
Parmag();
Writeasg(Cmchn,Mag("e"));
Purgoout(Cmchn);
Cadem=;False;
LSTpage=0;
LSTcount=LSTlinesperpage;
Ptr=Getcmd();
Chnl=Lstchn;
end
else begin
Chnl=Srcchn;
end;
until .Cadem do
begin
InitFDB(sixbit 'DSK', 0,
  case .Chnl=Binchn of
  set sixbit 'OBJ'; sixbit 'PII'; sixbit 'ABB'X 0 tes);;
Scanv=Filesca(Ptr,FDB); 
if not (Extsesn. (.Extension eq 0))
then Extension=sixbit 'ABB';
  case .Scanv(Retcode)++(-1) of
  set 0;
  Cmderr(Errs);
  Cmderr(PPNm);
  Cmderr(PPNm);
  Cmderr(PPNm);
  Cmderr(Mag("Invald switch specification"))
  tes;
  if .Scanv(Breach) eq ! "e"
  then Scanv(Breach) eq ! "e" 
  else if .Scanv(Breach) eq ! ";"
  then Scanv(Breach) eq ! ";"
select .Scanv(Breach) of
  next
  ";":exitselect begin
  if .Chnl eq Lstchn then Cmderr(Errs);
  if .Scanv then
  if not Inltchn(.Chnl,FDB) then Caddfail;
  if .Chnl eq Binchn then Chnl=Lstchn
  elseif .Scanv then Caddone;
  end;

  ";":exitselect begin
  if .Chnl eq Srcchn then Cmderr(Errs);
  if .Scanv then
```

if not Initchnl.Chnl,FDB) then Caddone;
Cr= exitselect begin
Cadeof=True;
if .Scanv
then (if .Chnl eqi Srcchn then
if not Initchnl.Chnl,FDB)
then Caddone;
elif .Chnl eqi Binchn then
Caddone;
Cadderr(Errm); end;
Aifelse: Cadderr(Msg('Illegal delimiter'))
test; false; end;
routine Punt(N) =
begin
external JobDoT,Jobca,SixlZ;
docr I from Hichn to 0 do
if .Chnlsta[1],Spoccl and not .Chnlsta[1],Inpio then
Purgeout(1); Cadswg(Msg('?M?J??PUNT!'));
if .JobDoT neq 0
then SixlZ(-1)
else (.Jobca());
value end;

! Source Input
!
comment ! Lread
!
Function: call the BLILIB routine READ, passing Srcchn as
! argument, and return the result, except that if the character read
! is a character of an SOS line number, return an error code of -3.
!
routine Lread =
begi
bind Vector Bufhed = Bufhd(Srcchn);
bind Bytptr = 1;
if Read(Srcchn) gtr 0 then
if @.Bufhd(Bytptr) then
Vreg=-3;
.Vreg;
end;
comment | OSEsrcchar
 |
| Function: get a character from the input buffer. Process line numbers, I/O errors, and end-of-file conditions correctly.
 |
global routine OSEsrcchar =
begin
 local Bitvec Char, InSOSno;
 InSOSno-ned;
 while (Char.Lread()) <= 0 do
 case .Char+3 of
 | 0 - Line number character
 | InSOSno-nonzed;
 | 1 - I/O error
 | begin
 | CdltsMsg(Msg('Input transmission error'));
 | return EOFchar;
 | end;
 | 2 - End-of-file
 | begin
 | Close(Srcchn);
 | Chnlstat(Srcchn,Wordf).Initstat(Srcchn);
 | 10bufp. Srcbufp;
 | if not Initchns(-1) then
 | return EOFchar;
 | end;
 | tes;
 | if .InSOSno ndq 0 and .Char eql HTab
 | then 0
 | else .Char
 | end;
 |
| Listing Output
 |
routine Listmsg(Str) =
begin
 Writemsg(Listchn,.Str);
 novalue
 end;

routine Page =
begin
 external Versid;
 bind Days(7x2) = split('Sunday',
 'Monday',
 'Tuesday',
 'Wednesday',
 'Thursday',
 'Friday',
 'Saturday');
end;
local Today;
LSTcount=0;
if not .Chnistat(Lstchn, TTYd) then
begin
  Today=Date();
  Lstmsg(Msg("?L;Rlgol 68 ' "));
  Lstmsg(Nptrtochars(VeraId));
  Lstmsg(Spcsm);
  Lstmsg(Nptrtochars(Days[2DayOfWeek(.Today)]));
  Lstmsg(Spcsm);
  Pdate(Lstchn, Today);
  Lstmsg(Spcsm);
  Ptime(Lstchn, -1);
  Lstmsg(Spcsm);
  Lstmsg(Cvtfile()));
  Lstmsg(Msg(" Page "));
  LSTheadpg=0;
  Cvtdecct(Nptrtochars(LSTheadpg), LSTpage - LSTpage-1, 3);
  Lstmsg(1pntochars(LSTheadpg));
  Lstmsg(CrFlm);
  Lstmsg(CrFlm);
  Purgeout(Lstchn);
end;
local Sav;
if (LSTcount, LSTcount, 0) gtr LSTlinesperpage then
begin
  Sav = scan(Foll);
  replace(Foll, 0);
  if .Chnistat(Lstchn, Specid) then
begin
Lstmsg(First);
Lstmsg(CrFlm);
if .Chnistat(Lstchn, TTYd) then Purgeout(Lstchn);
end;
if .Errdev neq 0 and not .Chnistat(Lstchn, TTYd) then
   Cmdmsg(First);
Object Output

global routine OSEobjword = novalue;

Error Message Output

global routine OSEerrmsg(N) =
begin
  external Errtxt;
  if .Errtxt(N) neq 0 then
    begin
      Outs('"');
      Outs (.Errtxt(N));
    end;
  novalue
end;

Initialization and Finalization

macro Initmem (duu) =
begin
  global GDLfreeo, GDLfreesq,Freesarea(3000);
  GDLfreeo = Freesarea(0,D); GDLfreesq = (Freesarea(3000)<0,D);
  novalue
end S;

routine Initprag =
begin
  GBLpragflags[0];
  GBLpragflags[Pragprom]=1;
  GBLpragflags[Pragwarn]=true;
  GBLpragflags[Praglist]=true;
  GBLpragflags[Pragobj]=true;
  novalue
end;

macro Inittransput(duu) =
begin
  until Inittrans(0) do Initprag();
  if .Chnstat[Lstchm,TTYd] then
    begin
      external Versid;
    end;
  novalue
end;

replace(Fill, .Sav);
end;
novalue
end;

Object Output

global routine OSEobjword = novalue;

Error Message Output

global routine OSEerrmsg(N) =
begin
  external Errtxt;
  if .Errtxt(N) neq 0 then
    begin
      Outs('"');
      Outs (.Errtxt(N));
    end;
  novalue
end;

Initialization and Finalization

macro Initmem (duu) =
begin
  global GDLfreeo, GDLfreesq,Freesarea(3000);
  GDLfreeo = Freesarea(0,D);
  GDLfreesq = (Freesarea(3000)<0,D);
  novalue
end S;

routine Initprag =
begin
  GBLpragflags[0];
  GBLpragflags[Pragprom]=1;
  GBLpragflags[Pragwarn]=true;
  GBLpragflags[Praglist]=true;
  GBLpragflags[Pragobj]=true;
  novalue
end;

macro Inittransput(duu) =
begin
  until Inittrans(0) do Initprag();
  if .Chnstat[Lstchm,TTYd] then
    begin
      external Versid;
    end;
  novalue
end;
macro Fintransput(dum) =
begin
  begin I from Hichn to 0 do
    if .Chnlist(I, .SpedI then
      Close(I);
  novalue
end$;

external Algo68;

Greet-0;
repeat
begin
  local Val;
  global GBLorrs, GBLwarns, GDBprogstart, GDBprogflags;
  Initmem();
  Initprog();
  Inittransput();
  if (Val-Algo68()) <= 0 then Punt(-Val);
  Fintransput();
end;

novalue
end
end$;

end
end$;
module A66S11(START = Start, NODEBUG) =
begin

Algol 68 Compiler Hydra Operating System Environment

This module contains most of the routines which comprise the
Hydra OSE for the Algol 68 Compiler. The routine Start is the
entry point for the Hydra load-and-go compiler system.

Compilation Initialization

The following functions must be performed:
- Ascertain which output files are to be created. Perform any
  setup required to make subsequent output actions work correctly.
  Set $Pragfile(Proglist) to true iff the listing file is to be
  written and $Pragfile(Progbj) to true iff the object file is to be
  written.
- Determine what text is to be used as the source input. Make any
  necessary connections to allow subsequent input actions to work
  correctly.
- Alter $Pragfile to reflect any requests for pragmas-controlled
  actions.
- Do anything else necessary for particular system.
- Set Fresho and Froehl.

Source Input ($SRCchar)
The routine $SRCchar returns the next character from the source input.
If no more input exists it returns EOFchar. The characters must be
encoded as 7-bit ASCII.

Listing Output ($LSTline)
$LSTline outputs a single line of text to the listing device and possibly
to the command output device as well. It takes three parameters, an
N-pointer to the first character of the line, an N-pointer to the character
immediately following the last character of the line, and an indication of
whether the line should be sent to the command output device (0 implies do
not send it).

Error Reporting ($Errmsg)
$Errmsg(n) is called to output any text message associated with error
number n.

Object Output ($OBJword)
$OBJword is called to output a word of object code.

Compilation Finalization

switches nolist;
require KERNAL.REQ (N81HY97);
require CDCODE.REQ (NA11HY97);
require STXPAG.REQ (NA11HY97);
require RSTSRA.REQ (NA11HY97);
require RTS.REQ (NA11HY97);
require PNCALL.REQ (NA10PH0D)
require TTCALL.REQ (NA10PH0D);
require B11PRM.REQ;
require COMMON.REQ;
require IONAC.R.REQ;
switches list:

undeclare
  SErrMsg,
  SListline,
  SOBJword,
  SSRCchar;

external
  Closedev,
  I0init,
  Opendev,
  Outascii,
  Out,
  Outcrlf,
  Versid,
  Glotog,
  Sline;

bind
  FatHarderr = FatUser-0,
  FatSignal = FatUser-1,
  FatObjbuf = FatUser-2,
  CM1chan = 0,
  CM0chan = 1,
  LPTchan = 2,
  RPSdata1 = 4,
  RPSdata2 = 5,
  RPSdata3 = 6,
  RPSio = 7,
  LNSs1x12 = 1,
  LNSyay = 2,
  LNSuser = 3,
  LNSport = 4,  x Parameter X
  LNSterm = 5,  x Parameter X
  LNSsource = 6,  x Parameter X
  LNSoptions = 7,  x Parameter X
  LNSobject = 8,  x Parameter X
  LNSSpagen = 13,
  LNSproccl = 16,
  LNSproccl2 = 17,
  LNSproccl3 = 18,
  LNSerrmsg = 19,
routine interrupt Herr =
  begin
  outs(’7G7G7G7G7G7G’)
  outs(’Hard Error at Compiler PC=’)
  outs(’Outoct(.OldPC); Eolerr();
  outs(’with ERRCODE=’)
  outs(’Outoct(.ERRCODE); Eolerr();
  if .Debcem neq 0 then
    Sixcmd(#1000);
  signal FatHarderr;
  end;

routine interrupt Hsig =
  begin
  local T;
  T:=RA;
  outs(’7G7G7G7G7G7G’)
  outs(’Signal #:’)
  outs(’Outoct(.T<0,15>); Outs(’at Compiler PC=#’)
  outs(’Outoct(.OldPC); Eolerr();
  outs(’with SIGDATA=#’)
  outs(’Outoct(.SIGDATA); Eolerr();
  if .Debcem neq 0 then
    Sixcmd(#1001);
  signal FatSig;
  end;

routine 10load(CPSslot) =
  begin
    if .CPSslot neq .10curpy then
begin
  IOcurpg-.CPSlot;
  SRLOAD (APSio,.CPSlot);
end;
novalue
end;

routine Punt(N) =
begin
  if .Qbcom neq 0 then
    Sirecmd(1787); 
    Outstr("PUNT!"); Outdec(.N);
  end;
novalue
end;

Source Input

global routine $SRCchar =
begin
  local Val;
  if .IOlnptr eq IOpage+20000 then
    (SRCpage=SRCpage+1; IOinptr=IOpage);
  IOload(SRCpage);
  Val=scanh(IOinptr); incp(IOinptr);
  if .Val eq 0 then
    if (Val=scanh(IOinptr) neq 0 then incp(IOinptr)
      else Val=EOFchar;
  .Val
end;

Listing Output

global routine Req0 SLSTline(First,Foll,Errdev) =
begin
  if .SPragflPrglst) or .Errdev neq 0 then
    begin
      if (LSTcount-LSTcount+1) gtr 54 then
        begin
          LSTcount=0;
          if .LSTchan neq CH0chan then
            begin
              Outasciz(LSTchan,Rng("?L");
              Outasciz(LSTchan,Versid);
              Cvtdec(LSTheadn,LSTpgno-LSTpgno+1,3);
              Outasciz(LSTchan,LSThead);
              Outerif(LSTchan);
              Outerif(LSTchan);
            end;
end;
end;
Outm(.LSTChan, .First, .Foll);
Outor(.LSTChan);
if .LST Chan neq CHDChan and .Errdev neq 0 then
begin
Outm(CHDChan, .First, .Foll);
Outor(CHDChan);
end;
end;
end;

// Object Output

bind
    Display = #2000, 
    Tranvect = #1000000, 
    Xpage = #120000, 
    Ypage = #140000;

macro SelectX(dum) = I0load(CPSxseg) $;
macro SelectY(dum) = I0load(CPSyseg) $;

global routine RegOBJword(Loc, Dtype, Dvalue) =
    begin
        local Doffset;
        bind Vector Basestable = uplt( 0, 0, 0, 0, Xpage, Ypage, Display, Tranvect )
        if .Loc gequ. Ybase
            then begin
                Loc = Loc—Ybase;
                SelectY(1);
            end
        else begin
            Loc = Loc—Xbase;
            SelectX(1);
        end
        if .Loc gequ #18000 then signal FatObject;
        Doffset=case .Dtype of
            set 
                .Dvalue;
                .Dvalue;
                .Dvalue;
                .Dvalue;
                .DvalueZadrinc = XbaseZadrinc;
                .DvalueZadrinc = YbaseZadrinc;
                .DvalueZadrinc =
                .DvalueZadrinc;
                I0page(Loc).Basestable(Dtype) = .Doffset;
                novalue
            end;
global routine RegO $Errmsg(N) =
    begin
    external Errtxt;
    DoLoad($PSerrmsg);
    if .Errtxt(N) neq 0 then
    begin
    Out(' - ');
    Outas(Errtxt(N));
    end;
    novalue
    end;

| Initialization and Finalization |

macro InItCusp(dum) =
    begin
    local $StrCBERRORTRAPS Traphandlers;
    local $Rights Restrict;
    Traphandlers($ERRPC)-ERR;
    Traphandlers($SIGPC)-Hsig;
    $SETLCB(0,Traphandlers,SCBERANTRAPS);  
    $COPY(LNSdata1,LNSprocdt1,$CPndata1);
    $COPY(LNSdata2,LNSprocdt2,$CPndata2);
    $COPY(LNSdata3,LNSprocdt3,$CPndata3);
    $RLOAD($Pndata1,$CPndata1);
    $RLOAD($Pndata2,$CPndata2);
    $RLOAD($Pndata3,$CPndata3);
    $CPLOAD(0,
        $CPnxtmpl,$PATH(LNSnxtmpl),,,
        $CPnxtmp,
        Restrict($StrCBERRORTRAPS));
    incr I from 1 to SCLEN(LNSnxtmpl) do
    begin
    $GETCAP($H$Htmp,$Path(LNSnxtmpl,,I));
    $PUTCAP($H$Htmp,$H$Htmp,Restrict);
    $CPLOAD(0,<$CPsource-I,,I,$CPsource>);
    $DELETE($H$Htmp);
    end;
    novalue
    end;

macro InitMem(dum) =
    begin
    csell global='FREE.O';
    global Frearea;
    bind Frearea=10page;
    $Free=Frearea;
    $Freei=Frearea;
    $Freei=Frearea;
routine Deswitches =
begin
stacklocal Len;
Len=SDL LENGTH(LNSoptions);
incr 1 from 2 to .Len do
begin
stacklocal Sw;
SGETOPT(Sw,LNSoptions,1,1);
if 8 less .Sw and .Sw less 17 then
  case .Sw of
  set
  X DEBUD X Debcom-onned;
  X GHOST X SPragfl(Pragnghost)-false;
  X LISTING X SPragfl(Praglist)-true;
  X LOWER X SPragfl(Pragstrop)-3;
  X MACH X SPragfl(Pragmach)-true;
  X NAILED X SPragfl(Pragnnonked)-false;
  X NODEREG X Debcom-onned;
  X NOGHOST X SPragfl(Pragnghost)-true;
  X NOLISTING X SPragfl(Praglist)-false;
  X NONINCH X SPragfl(Pragmach)-false;
  X NONNAILED X SPragfl(Pragnnonked)-true;
  X NONWARNINGS X SPragfl(Pragwarn)-false;
  X POINT X SPragfl(Pragstrop)-1;
  X RES X SPragfl(Pragstrop)-0;
  X UPPER X SPragfl(Pragstrop)-2;
  X WARNINGS X SPragfl(Pragwarn)-true
  end;
end;
novalue
end;

macro Initprags(dum) =
begin
SPragfl=0;
SPragfl(Pragstrop)=1;
SPragfl(Pragwarn)=true;
SPragfl(Pragmac)=true;
SPragfl(Pragobj)=true;
SPragfl(Praglist)=true;
Deswitches();
novalue
end#;

routine Inittransput =
begin
10init(LNSpace);  
Opendev(CMOchan,CMObuf(0),88,3);  
SGETOPT(LSTdev,LNSoptions,1,1);  
if .LSTdev eq 8  
  then SPragfl(Praglist)=false  
  else .LSTdev eq 2  
  then begin  
    Opendev(LPTchan,LPTbuf(0),136,9);  
end;
LSTchan=LPTchan;
end;
Outascii(CMOchan,Versid);
Outascii(CMOchan,Msg(" Compiler Starting"));
Outcrlf(CMOchan);
novalue
end;

routine Fintransput =
begin
if .LSTchan eq LPTchan then
Closedev(LPTchan);
Closedev(CMOchan);
novalue
end;

routine AG8 =
begin
local Val;
external Algol68;
Initcusp();
Initmem();
Initprags();
Inittransput();
if .Debcom neq 0
then Sixedad(8)
else Global--1;
if (Val-Algo(68)) eq 8
then Val=Zadrinc.$Prog=$page
else begin
if .Val las 0 then Punt(-.Val);
Val=0;
end;
SPUTDATA(LHSobject,Val,1,1);
Fintransput();
$BRERL(LHSterm);$.Val
end;

routine Start =
begin
external Init612,Ret612;
csect global="CL00.6";
global $Errors,$Freeos,$Freeel,$Progl,$Proqst,$Warns;
Init612(LHSport,0,0,1,1);
Ret612(AG8(),0,0)
end;

end
$ludew
macro
    Mode = Oblock $,
    XMode = 1 $,

    Mdv = 1,Wordf $,
    Md10 = 1,RH $,
    Mdsein = 1,Wordsiz-8,1 $,
    Mddone = 1,Wordsiz-7,1 $,
    Mddeprec = 1,Wordsiz-6,1 $,
    Md1special = 1,Wordsiz-5,1 $,
    Mddrecu = 1,Wordsiz-4,1 $,
    Md1addressed = 1,Wordsiz-3,1 $,
    Md10 = 1,Wordsiz-2,1 $,
    Md1pile = 1,Wordsiz-1,1 $,

    Md1prefad = 2,Wordf $,
    Md1unddata = 2,Wordf $,

   Md1unocnt = 2,Wordf $,
    Md1unocnt(n) = 3+(n),Wordf $,

    Md1pres = Md1prefad $,
    Md1precnt = 3,Wordf $,
    Md1preprn = 4,Wordf $,
    Md1preprn(n) = 4+(n),Wordf $,

    Md1rad = Md1prefad $,
    Md1vted = Md1prefad $,

    Md1round = 2,Wordf $,
    Md1roundcnt = 3,Wordf $,

    Md1strSDB = 2,Wordf $,
    Md1strian = 3,Wordf $,
    Md1strcnt = 4,Wordf $,
    Md1strfields = 5,Wordf $,
    Md1strfield(n) = 5+2e(n),Wordf $,
    Md1strflex(n) = 6+2e(n),Wordf $;

! Definitions of Md1 values for modes.

macro
    Md1vval(uniq,pi,dras,spc,io) =
        (pi)t(Wordsz-1) + (io)t(Wordsz-2) + (dras)t(Wordsz-3)
        + (spc)t(Wordsz-5) + (uniq) 5;
bind

Pile = 1, Simple = 0,
Dressed = 1, Undressed = 0,
Special = 1,
lo = 1, NoLo = 0,

Daproc = If(Qurediz-0);

bind Constbv

Advint  = Advval( 0, Simple, 0, 0, lo );
Advint  = Advval( 1, Pile, Undressed, Special, lo );
Advreal = Advval( 5, Pile, Special, lo );
Advchar = Advval( 4, Simple, 0, lo );
Advbits = Advval(5, Simple, 0, lo );
Advbytes = Advval (6, Simple, 0, lo );
Advstring = Advval(7, Pile, Dressed, 0, lo );
Advbool = Advval(8, Simple, 0, lo );
Advvoid = Advval(9, 0, 0, 0, lo );
Advtrip = Advval(10, 0, 0, 0, lo );
Advjump = Advval(11, 0, 0, 0, 0);
Advnit = Advval (12, 0, 0, 0, 0); Advout = Advval (13, Pile, Dressed, 0, 0); Advin = Advval (14, Pile, Dressed, 0, 0);
Advoutb = Advval (15, Pile, Dressed, 0, 0);
Advnumber = Advval (16, Pile, Dressed, 0, 0);
Advcons = Advval (18, Pile, Dressed, 0, 0);
Advunionof = Advval (19, Pile, Dressed, 0, 0);
Advboola = Advval (20, Simple, 0, 0, 0); Advboolb = Advval (21, Simple, 0, 0, 0);
Advproc = M-Advval (72, Pile, Dressed, 0, NoLo);
Advval = Advval (23, Pile, Dressed, 0, NoLo);
Advstruct = Advval (24, Pile, Undressed, 0, NoLo);
Advcons = Advval (25, Pile, 0, 0, NoLo);
Advvent = Advval (26, Pile, Dressed, 0, NoLo);
Advcode = Advval (27, Simple, 0, 0, 0);
Advnsimple = Advval (28, Simple, 0, 0, NoLo);
Advnspile = Advval (29, Pile, Dressed, 0, 0, NoLo);

macro

Callmode(H) = (Nonprint(H) eql 1) $,
Condmode(H) = (H(FD10) eql Advcode(H(FD10)) $,
Eventmode(H) = (H(FD10) eql Advevent(H(FD10)) $,
Hiptmode(H) = (H(FD10) gtr 0) $,
Inputmode(H) = (H(FD10) leq Advreal(H(FD10)) $,
Inummode(H) = (H(FD10) gtr Advcode(H(FD10)) $,
Procmode(H) = (H(FD10) eql Advproc(H(FD10)) $,
Itmode(H) = (H(FD10) eql Advrel(H(FD10)) $,
Roundmode(H) = (H(FD10) eql Advround(H(FD10)) $,
Structmode(H) = (H(FD10) eql Advstruct(H(FD10)) $,
Unddefinedmode(H) = (H(FD10) eql Advabsent(H(FD10)) $,
Unionmode(H) = (H(FD10) eql Advunion(H(FD10)) $,
Unitmode(H) = (Tuned(H) gtr 0) $,
Widenable(H) = (H(FD10) eql Advstring(H(FD10)) $,
Widenatoroun(H) = (H(FD10) eql Advbits(H(FD10)) $, I Valid only if Widenable(H) is true
\begin{verbatim}
Samesize(X1,X2) = ( (X1) eqv (X2) )

external
RegO  Valuelength,
RegO Tarith,
RegO Tchars,
RegO Thip,
RegO Tsize,
RegO Tnonplain,
RegO Tunitsd,
Tunion,
RegO TXsize;
bind
Xint    = 1,
Xlint   = 2,
Xreal   = 3,
Xrreal  = 4,
Xcompl  = 5,
Xicompl = 6,
Xmaxsizint  = 2,
Xmaxszreal = 4,
Xchar   = 1,
Xstring = 2,
Xchars  = (Xchare2 + Xchar) -2,
Xskip   = 1,
Xjump   = 2,
Xnihil  = 3,
Xout    = 1,
Xin     = 2,
Xoutb   = 3,
Xinb    = 4,
Xnumber = 5,
Xrows   = 6;

! END OF HOSTRAC.REQ
\end{verbatim}
module Lexan =
begin

Lexical Analyzer

The lexical analyzer converts a stream of characters into a stream of lexemes representing tokens of the ALGOL 685 language. The stream of characters is the source program and is supplied one character at a time by the system-dependent routine OSEsrcchar. The next lexeme of the program is obtained from the analyzer by a call on Lexan. A description of the format and possible values of lexemes may be found in LXSTRC.REQ.

A more specific list of the functions performed by the lexical analyzer is the following:
- Recognize tokens in the current stropping convention
- Construct lexemes
- Output program listing
- Ignore comments
- Process pragmas

A source program may be represented in any of four stropping conventions. The current default is the POINT convention in which bold tokens (i.e. bold-tags and symbols represented by bold character sequences) are indicated by a prefix stropping character. This stropping character may be either a point (.) or an apostrophe ('). The other conventions available are UPPER, LOWER and RES. The use of the stropping character is permitted in all of these. The occurrence of a stropping followed by an alphabetic character always forces the following alphanumerics sequence to be bold.

The UPPER convention uses upper case letters to indicate bold letters, while the LOWER convention uses lower case letters. A stropping may be used to override this as described above.

In the RES convention, typographical display features may no longer appear between the marks of a tag or the symbols of a denotation.
A sequence of alphanumerics characters surrounded by disjunctors (non-alphanumerics characters or typographical display features) represents a bold symbol, if one exists with that particular spelling; otherwise it represents a tag.
In RES stropping the first occurrence of a user-introduced bold-tag must be explicitly stropped.

Known Nonconformance with the Harman-Boom Report
- There is no intimidation character. In particular, there is no representation of the worthy character underscore.
- Upper and lower case letters may be intermixed following a stropping in UPPR (and LOWER); thus, "Begin is always allowed.
- PAGE pragmat-item is not implemented.
I - Apostrophe-image symbol is not implemented.
I - The RES convention is different. It probably does not even conform to the Report because of the restriction on the use of typographical display features. Also, applied occurrences of bold-tags are reserved.
I - $ is permitted as the first character of a tag.

switches no list;
require SYSPRM.REQ;
require COMMON.REQ;
require IOHRC.REQ;
require LISPKG.REQ;
require ERRCON.REQ;
require ERREX.REQ;
require LXTABS.REQ;
require LXTARC.REQ;
require MDTRC.REQ;
require LXVCN.REQ;
require LXVSY.REQ;
require LXTBL.REQ;
require MDTBL.REQ;

switches list;

external
Errchar,
Errxptr,
Errnonblank,
Errptr,
Lineno,

Pblock Faptr;  I Pointer to first location in free area
forward
InitLx,  % global %
Reg0 Lxerr,
Outsrc,
Nexch,
Reg0 Nxtch,
Reg0 Scanch,
TieABC,
Tboid,
Thread,
Putchr,
Putcvt,
Uppercase,
Hashin,

Reg0 Dupreg,
Lxscan,
Lx,
Errrchar,
Getprmdn,
Getstrgn,
Reg0 Scantax,
Hashhold,
Gettax,
Getbold,
Endoff,
macro
Srcstat = Srcbuf[G];  // Nptr

bind
Srcline = Srcbuf[Iptrndx(2)],  // Iptr
Srcext = Srcbuf[Iptrndx(8)],  // Iptr
Srcend = Srcbuf[Iptrndx(Charbufsize+1)],  // Iptr
Errtext = Errbuf[Iptrndx(8)];  // Iptr

comment InitLx

// Functions: Perform post-compilation initialization required by
// the Lexical Analyzer.

global routine InitLx =
begin
    local.Ptr;
    Index-Control;
    Inpragment-zad;
    Lookahed-0;
    Longcnt-0;
    Ptrl-1ptrtochars(Errbuf);
    until .Ptr eql Errtext do replace(Ptr,"");
    Errptr-Errexprt-Ptr;
    Errchar-"";
    Errnontblank-zad;
    Ptrl-1ptrtochars(Srcbuf);
    replace(Ptr,"";
    until .Ptr eql Sctext do replace(Ptr,"";
    Sctpstr-Ptrl;
    Lineno-1;
    Zerocnt(H,Hsize);
    Faptr=Lextab-0,0;
    until .Faptr[Link] eql Minus[Link] do begin
        if .Faptr[Lv] eql Lvtag then
            Faptr[Exp]=0;
        ...
Hashin();
end;
novalue
end;

comment i Lxerr(N)
|
| Functions: Print error message unless currently processing inside
| a program.
| Value: The Error lexeme.
| Inputs
| N — Error code
| Inprogram — nonzed iff scanning program
| routine Lxerr(N) =
| begin
| if .Inprogram eq 8 then
| Error(N,); Lxerror end;

comment i Outsrc
|
| Functions: Output a line of source listing on the listing device.
| If an error occurred in the line or the line was ignored due to a
| previous error, then a line of error indication is also output.
| If an error occurred in the line, then Errdev will be nonzed and thus
| all output will go to the error device also.
| routine Outsrc =
| begin
| Cvtides(Srctext, .Lineno,3); Lineno-=.Lineno+1;
| Srcptr=ItolNptr(.Srcptr); incp(Srcptr);
| OSELstline(Nptrtochars(Srctext), .Srcptr, .Errdev);
| Srcptr=Srcstxt;
| if .Errnonhiant eq 0 then
| begin
| Errptr=ItolNptr(.Errptr);
| incp(Errptr);
| OSELstline(Nptrtochars(Errbuf), .Errptr, .Errdev);
| if .Errchar eq '"' then
| Errnonhiant+=ed;
| Errdev+=ed;
| end;
| Errptr=Errinptr(.Errtext); Screte=Srcstxt;
| novalue
| end;
Function: Get a non-control character from the input stream.

```
routine Nextch =
  begin
    local C;
    do C .OSEarchar() until .Chartype(C) neq Cctrl;
    .C
  end;
```

bind

- Skipnone = EOL,
- SkipEOL = Space,
- Skipspaces = Errch,
- Skipinprag = Letter;

Function: Get the next acceptable character from the source input.
Level Is used to indicate which characters are acceptable. Nextch always uses a new input character as the first character to consider; Scanch starts with the current character.

```
routine Ray0 Nextch(Leve1) =
  begin
    do begin
      if .Index eqi EOL then Outsrc();
      if .Lookahead neq 0 then (Char=Lookahead; Lookahead=0)
        else Char=Nextch();
      Type=.Chartype(Char);
      Index=.Typo(index);
      if .Index eqi EOF then (if .Srcptr neq Srcstart then Outsrc())
        elsif .Index grt EOL then begin
          if .Index eqi Point then
            if (Lookahead=Ray0) geq "A" then begin
              Type=Chartyp;
              Index=Strrep;
            end;
          if .Srcptr eqi Srcend then Outsrc();
          replace(Srcptr,.Char);
          replace(Srcptr,.Errchar);
        end;
      end;
    end while .Index leq .Level;
  novalue
  end;
```
routine RngO Scanch(Level) =
begin
  if .Index < .Level then Nextch(.Level);
  novalue
end;

comment I Tdigit,Tletter,Tupper,Tlower,Tdigit,Tdigitper,
| Tdigit,Thexit,TletterR,Tpow10,TletABC,Tbold,
| Tbreak (Bold)
| Value: true if the current character is --
| (Tdigit) a digit
| (Tletter) a letter
| (Tupper) an upper case letter
| (Tlower) a lower case letter
| (Tdigit) a digit or letter
| (Tdigitper) a digit or upper case letter
| (Tdigitper) a digit or lower case letter
| (Thexit) a hexadecimal digit
| (TletterR) a letter R
| (Tpow10) a power of ten choice
| (TletABC) a non-bold letter symbol
| (Tbold) a possible first character of bold tag
| (Tbreak) not allowed in tag or bold-tag

macro
  Tdigit (dum) = (.Type(Tpdigit)) $,
  Tletter (dum) = (.Type(Tplettar) neq 0) $,
  Tupper (dum) = (.Type(Tpupper)) $,
  Tdigit (dum) = (.Type(Tpdigit) neq 0) $,
  Tdigitper (dum) = (.Type(Tpdlgper) neq 0) $,
  Thexit (dum) = (.Type(Tpexite) and (TletABC() or Tdigit()) $,
  TletterR (dum) = (TletABC() and Uppercase() eqi "R") $,
  Tpow10 (dum) = (.Char eqi "0") or
    (TletABC() and Uppercase() eqi "E") $;

routine TletABC =
  case .GBL pragflags(Prgstrop) of
  set
    bool(Tletter());
    bool(Tletter());
    bool(Tlower());
    bool(Tupper())
  end;

routine Tbold =
  if .Index eqi Strop
    then true
  else case .GBL pragflags(Prgstrop) of
  set
    % RES  % bool(Tletter());
    % STROP % false;
    % UPPER % bool(Tupper()));
macro Tbreak (Bold) =
  begin
  case .GBDpragflags[Prsprog] of
    set
      % RES % not Tdigit() ;
      % STROP % not Tdigit() ;
      % UPPER % if (Bold) neq 0 then not Tdigit() else not Tdigit() ;
      % LOWER % if (Bold) neq 0 then not Tdigit() else not Tdigit() ;
    not;
  end;
end;

comment Putchr and Putcvt

| Function: Store character in L padding Table entry under construction
| at top of freearea. Update character count. Check if there is sufficient
| space left in freearea. Putcvt, in addition, first converts any
| lower case letter to the corresponding upper case letter.
| Inputs
| Symptr — Nptr to character string in new L padding Table entry
| Symcnt — count of characters in current symbol
| Outputs
| Symptr — updated after character is stored
| Symcnt — incremented by one
| Routine Putchr =
begin
  if modulo(Symcnt,2079) eq 0 then
    begin
      Storatot((Symptr<Addr>));
      (.Symptr)<Nordfo>B;
      end;
      replaceN(Symptr,.Char);
      incp(Symptr);
      Symcnt « Symcnt+1;
      novalue
      end;
Routine Putcvt =
begin
  Char « Uppercase();
  Putchr();
  novalue
  end;

comment Uppercase

| Function: Compute the upper case equivalent of the current input character.
| Routine Uppercase =
| if Tlower() then .Char and (not #48) else .Char;
comment ! Hashin

Function: Search Lexeme Table for lexeme sitting at start of freearea.
If lexeme is already in table, then return pointer to this
old lexeme. If it is not in the table and Noenter is not set and
we are not inside a pragman, then enter the lexeme in the table
and return a pointer to the new lexeme. If lexeme is not found and
a new entry is not made then return zero.

global routine Hashin =
begin
local Total,Ptr;
Total-0;
Ptr=Ptrtochar(Faptr[Lxsym]);
door I from .Faptr[Lxcount]+3 to 0 do
begin
Total=.Total+scanN(Ptr);
incp(Ptr);
end;
Total=.Total and HTmask;
do Total=.Total-HTsize until .Total Is 0;
Total=.Total-HTsize;
Noenter=.Noenter or .Inpragma;
Find(HTL.Total),HTD(.Faptr[Lxcount]+1);
end;

comment ! Doprag(N)

Function: Carry out the semantics of the pragman specified by N.

macro Doprag(N) =
begin
case (N) of
set
! RES, POINT, UPPER, LOWER
Newstrip-(N);
Newstrip-(N);
Newstrip-(N);
Newstrip-(N);

! WARNINGS and NOWARNINGS
GBLpragflags[Prwarn]=true;
GBLpragflags[Prwarn]=false;

! LISTING and NOLISTING
GBLpragflags[Prlist]=true;
GBLpragflags[Prlist]=false;

! PAGE
les;
novalue
endS;
comment 1 Lxscan

1 Function: Scan a token from the input and return its lexema.
1 A token consists of an optional fragment (pragma or comment) followed
1 by a symbol.
1
1 global routine Lxscan =

begin
    label L;
    local Lexema Lex;
    while begin
        SrcstdL', N;
        LexatwO;
        .Lex[Lxv] eql 0
    end do begin
        local Neustrop;
        Inpragma'='Scratch'.Lex[Lxp];
        Neustrop'='.GBLpragmaflags[Pragmaoper];
        repeat begin
            local Lexema Lex2;
            Scanch(SrpinPrag);
            Lex2=Lx();
            if .Lex2 neq Lexerror then
                if .Lex2 eql Lexstop
                    then (Error(ELx2); exitloop)
                elseif .Lex2 eql .Lex
                    then exitloop
                elseif .Lex2[Lxv] eql Lxvpragma and .Inpragma eql "p"
                    then Doprag(.Lex2[Lxp])
                end;
            Inpragma=xed;
            GBLpragmaflags[Pragmaoper]=Neustrop;
            end;
        Lxselect .Lex[Lxv] of
            nest
                Lxvlong; exitL Longcont'=.Longcont+1;
                Lxvshort; exitL Longcont'=.Longcont-1;
            Allsize; Longcont=0
        test;
        .Lex;
    end;

comment 1 Lx

1 Function: Scan a symbol from the input.
1 Value: Lexema for the symbol.
1
1 routine Lx =

begin
    local Pblock Lex;
    bind LxRoutP = uplfl:
        0, 1 Control
        0, 1 EOL
do begin
  Scanch(Skipspaces);
  Errixptr.Errptr;
  S: {ent-0;
  Symstr=Matrchar(Faptr(Lsys));
  Lex.(LexRoute(.Index))();
  end;
  until .Lex neq Lexerror or .Inpragmnot neq 0;
  .Lex;
end;

comment ! Errorchar
!
| Function: Deal with situation where illegal character occurs in the source.
| Value: Error lexems.
|
routine Errorchar = (Lexerr(ELxR); Nextch(Skipnono); Lexerror);

comment ! Getprimden
!
| Function: Scan a primitive denotation from the input.
| Value: Lexems for the denotation.
|
routine Getprimden =
begin
  local State, Mode N;
  external Lengthen, Cvib,Cvt11,Cvtr,Cvtr,Cvtrl;
  bind Vector Cvtinv = uplilt(Cvib,Cvt11,Cvtr,Cvtr,Cvtrl);
  State(1,Index eqi Point then 1 else 0);
  repeat
    begin
      Putcvt();
      Nextch(ii :GBLpragflags(Prgstrop) eqi 0 then Skipnono else Skipspaces);
      case .State of
        set
          0: scan digits
          if .Index eqi Point
              then State=1
          elf TletterR();
then State-8
   elsif TpowlO()
       then (Char="E"; State-3)
   elsif not Tdigi()  
       then (M-Float; exitloop)
   
   ! 1: fixed-point-numeral must follow point in fractional-part
   if Tdigi()
       then State-2
       else return Lxarr(ELx4)
   
   ! 2: scan digits of fractional-part
   if TpowlO()
       then (Char="E"; State-3)
   elsif not Tdigi()
       then (M-Float; exitloop)
   
   ! 3: check for plusminus in exponent-part
   if .Index eql Pismain
       then State-4
   elsif Tdigi()
       then State-5
       else return Lxarr(ELx4)
   
   ! 4: fixed-point-numeral must follow plusminus
   if Tdigi()
       then State-5
       else return Lxarr(ELx4)
   
   ! 5: scan fixed-point-numeral in exponent-part
   if not Tdigi()
       then (M-Float; exitloop)
   
   ! 6: digits must follow letter-r in bits-denotation
   if Thexit()
       then State-7
       else return Lxarr(ELx4)
   
   ! 7: scan digits in bits-denotation
   if not Thexit()
       then (M-Float; exitloop)
   
end;
M-Length([N, Longcnt])
replaceN(Symptr, 0)
if (.Cvtln(Tarit([N]))) gtr 8 then return Lxarr(ELx4)
Faptr[Lxdenndt]-N;
Faptr[Lxv]=Lxvtialen;
Faptr[Lxp]-8;
Faptr[Lxdata]=ValueLength([N]+1;
Faptr[Lxtoken]=TxDenot;
Hashin()
end;

comment I Getstryned
Function: Scan a string denotation from the input.

Value: Lexeme for the string denotation.

```
routine Getstrgden =
    begin
        external Cutstrg;
        Sympt.+Npt.+tochar(Faptr(Lxstrgrp));
        repeat begin
            Sreak-"S";
            Nextch(SkipEOF);
            if .Index eqi EOF
                then return Lxarr(ELxL)
            else
                begin
                    Srcstch~ a;
                    Nextch(Skipnone);
                    if .Index sql Quote
                        then Putchr()
                    else
                        begin
                            Scanch(Skipspaces);
                            if .Index sql Quote
                                then exitloop;
                            end
                        end;
            Cvtstrg(Symcnt);
            if .Symcnt eqi 1
                then (Symcnt=0; Faptr(Lxdenrep)=Adchar; Faptr(Lxdenrep)=Faptr(Lxstrgrp))
                else (Faptr(Lxdenrep)=Adstring; Faptr(Lxdenrep)=Symct);
            Faptr(Lxw)=Lxstrgden;
            Faptr(Lxpl)=0;
            Faptr(Lxcount)=divide(.Symcnt+3eZchswrd-1,Zchswrd);
            Faptr(Lxtoken)=TtDenot;
            Hashin()
            end;
    end;
```

comment I Scanstax(Bold)

Function: Scan characters contained in a tag or bold-tag.

Inputs
   Bold - nonced iff scanning bold-tag

```
routine Rug0 Scanstax(Bold) =
    begin
        do begin
            Putctv();
            Nextch(if .Bold sqi 0 then Skipnone else Skipspaces);
        end
        until Tbreak(.Bold);
    end;
```
comment // Hashbold

// Function: Scan a bold-tag from the input.
// Value: Lexeme for the bold-tag.
routine Hashbold =
begin
Scantax(nonced);
Faptr(Lxv)=Lxvtab;
Faptr(Lxtoken)=TkBold;
Hashin();
end;

comment // Gettax

// Function: Scan a TAX-symbol from the input. In RES convention the symbol scanned may be a TAB-symbol, otherwise it is a TAG-symbol.
// Value: Lexeme for the symbol.
routine Gettax =
begin
if Tbold()
then begin
  local Val;
  if .CBLpragflagstPrag.lop=O then
    Noenter=nonced;
  if (Val=Hashbold(0)) neq O then
    return .Val;
  end
  else
    Scantax(nonced);
  Faptr(Lxv)=Lxvtag;
  if .Inprag=ent eqi acb then
    Lexerror
  else begin
    Faptr(Lxtoken)=(if .Inprag=ent eqi O then TkTag else TkPraglene);
    Hashin();
  end
end;

comment // Getbold

// Function: Scan a bold-tag from the input.
// Value: Lexeme for the bold-tag.
routine Getbold =
begin
local Val,Savstrop;
Nextch(Skipnonmo); if not Tletter(0) then
comment ! Endoffile
!
! Function: Return STOP lexeme to indicate source end-of-file has occurred.
!
routine Endoffile = return Lexstop;

comment ! Getopr
!
! Function: Scan an operator from the input.
!
! Value: Lexeme for the operator or Error lexeme.
!
routine Getopr =
begin
local Lxptr,S,Olds;
Lxptr=Lxerror;
S=0;
do begin
Olds=S;
if .Char eqi .Opchtable(.S,0char)
then begin
Nextch(Skipnona);
Lxptr=.Opchtable(.S,0tral);
if .Opchtable(.S,0next) then S=.S+0tral;
end
else begin
S=.S+.Opchtable(.S,0tral);
end;
until .S eqi .Olds;
if .Lxptr eqi Lxerror
then Lxerr(ELx3)
else Opixtable(Lxptr)<0,0>
end;

global routine Lxline =
begin
Scanchs(Skipspaces);
novalue
end;
end
eludes
module SRtabs
begin
  Semantic Tables
  
  This module defines the entries initially contained in the NODE Table, OPERATOR Table and SYMBOL Table.

  Kernel call tables
  
in - XX<Z ... Z<XX
  out - XX<Z ... Z<XX

  Hydra tables (other than Kernel call)
  
in - XX<Z ... Z<XX
  out - XX<Z ... Z<XX

  switches nolist;
  require SYSPRN.REQ;
  require COMMON.REQ;
  require LISPSEQ.REQ;
  require LXXCOM.REQ;
  require LXXSY.REQ;
  require LXXVAL.REQ;
  require EXIT.REQ;
  require MODSTRC.REQ;
  require STSTRC.REQ;
  require OPSTRC.REQ;
  require LXXTABL.REQ;
switches list;

macro empty = 0, 0 $;

Defstid (offset, md, lx) =
  0, Oadrincupilit (0, lx), 0, Hfwords (Stbtype (Stidty, Stbdefid), 0),
  (offset + 2xiz1base + 2xiz1EDbase + lp), 0, md 5,

Defstop1 (prio, std, lx) =
  0, Oadrincupilit (0, lx), 0, Hfwords (Stbtype (0, Stbdefprio), 0),
  std, 0, prio 5,

Defstop2 (prio, std) =
  0, 0, 0, Hfwords (Stbtype (0, Stbdefprio), 0),
  std, 0, prio 5,

Defdonint (val) =
  Lxxm (Lxvprnden, val), Hfwords (2, TdDenot), Mdint, val 5,

Defdog (len, str) =
  Lxxm (Lxvtag, 0), Hfwords ((len+Qchsurdf-1)/Qchsurfd, TdTag), str 5,

Defbold (lv, lp, len, str) =
Defpragma(int, int, str) =
    Lexm(Lxvpragma), Hiwords((len+Ochword-1)/Ochword, TkPragma), str $,$

Defop(int, str) =
    Lexm(Lxvopr, O), Hiwords((len+Ochword-1)/Ochword, TkBold), str $,$

Defcode0(yield) = Advcode+Doproc, yield, 0 $,$
Defcode1(p1,yield) = Advcode, yield, 1, p1 $,$
Defcode2(p1,p2,yield) = Advcode, yield, 2, p1, p2 $,$

Defproc0(yield) = Advproc+Doproc, yield, 0 $,$
Defproc1(p1,yield) = Advproc, yield, 1, p1 $,$
Defproc2(p1,p2,yield) = Advproc, yield, 2, p1, p2 $,$
Defproc3(p1,p2,p3,yield) = Advproc, yield, 3, p1, p2, p3 $,$
Defproc4(p1,p2,p3,p4,yield) = Advproc, yield, 4, p1, p2, p3, p4 $,$
Defproc5(p1,p2,p3,p4,p5,p6,yield) = Advproc, yield, 6, p1, p2, p3, p4, p5, p6 $,$

Defraf(md) = Advref, md $,$
Defrow(md,rcnt) = Advrow, md, rcnt $,$
Defunion2(m1,m2) = Advunion2, 2, m1, m2 $,$
Defstruct2(m1,m2,m3,m4) = Advstruct, 0, 8, 2, m1, m2, m3, m4 $,$

binding dummy = 0;

dataarea(HdD):

gbname(Hdint, Mdvint),
gbname(Hdint, Mdvint),
gbname(Hdbits, Mdvbits),
gbname(Hdbytes, Mdvbytes),
gbname(Hdreal, Mdvreal),
gbname(Hdreal, Mdvreal),
gbname(Hdbool, Mdvbool),
gbname(Hdchar, Mdvchar),
gbname(Hdchar, Mdvchar),
gbname(Hdstring, Mdvstring),
gbname(Hdfile, Mdvfile),
gbname(Hdname, Mdvname),
gbname(Hdvoid, Mdvvoid),
gbname(Hdint, Mdvint),
gbname(Hdint, Mdvint),
gbname(Hdjump, Mdvjump),
gbname(Hdnil, Mdvnil),
gbname(Hdout, Mdvout),
gbname(Hdin, Mdvint),
gbname(Hdoubt, Mdvoutb),
gbname(Hdin, Mdvint),
gbname(Hdnrnumber, Mdvnumber),
gbname(Hdrown, Mdvrown),
gbname(Hdbnode, Mdvnode),
gbname(Hdabsent, Mdvabsent),
gbname(Hdout, Mdvproc)
data()

gblname(Mdcompl, Defstruct2(Mdreal,Lexre,Mdreal,Lexim)),
data()
gblname(Mdcompl, Defstruct2(Mdreal,Lexre,Mdreal,Lexim)),
data()
gblname(Rowbool, Defrow(Mdbool, 1)),
data()
gblname(Rowchar, Defrow(Mdchar, 1)),
data()
locname(Rowout, Defrow(Mdout, 1)),
data()
locname(Rowin, Defrow(Mdin, 1)),
data()
locname(Rownutb, Defrow(Mdoutb, 1)),
data()
lcnnames(Rownutb, Defrow(Mdoutb, 1)),
data()
locname(Rowinb, Defrow(Mdinb, 1)),
data()
gblname(Prbnd, Defprocl(Mdbnds)),
data()
locname(Prbnd, Defprocl(Mdbnds, Mdvoid)),
data()
locname(Prziv, Defprecl(Mdint, Mdint, Mdvoid)),
data()
locnnames(Prziv, Defprecl(Mdint, Mdint, Mdvoid)),
data()
locname(Refint, Defref(Mdint)),
data()
locname(Refint, Defref(Mdint)),
data()
locname(Refstring, Defraf(Mdstr)),
data()
locname(Refstring, Defraf(Mdstr)),
data()
gblname(Rcrlv, Defprocl(Reflla, Stdvoid)),
data()
locname(Rcrlv, Defprocl(Reflla, Stdvoid)),
data()
locname(Rcrlv, Defprocl(Reflla, Stdvoid)),
data()
locname(Rcrlv, Defprocl(Reflla, Stdvoid)),
data()
locnnames(Rcrlv, Defprocl(Reflla, Stdvoid)),
data()
locnnames(Rcrlv, Defprocl(Reflla, Stdvoid)),
data()
locname(Previn, data(0)), Dofproc1(Rainin, Mvoid)), 1 Read
locname(Prevob, data(0)), Dofproc1(Rainout, Mvoid)), 1 Write bin
locname(Previb, data(0)), Dofproc1(Rainin, Mvoid)), 1 Read bin
locname(Prevfout, data(0)), Dofproc2(Rainout, Rainout, Mvoid)), 1 Put
locname(Prevvin, data(0)), Dofproc2(Rainfile, Rainin, Mvoid)), 1 Get
locname(Prevup, data(0)), Dofproc2(Rainfile, Rainout, Mvoid)), 1 Putbin
locname(Prevfg, data(0)), Dofproc2(Rainfile, Rainin, Mvoid)), 1 Getbin
locname(Prevfrs, data(0)), Dofproc2(Rainin, Rainout, Mvoid)), 1 Make term
locname(Procon, data(0)), Dofproc2(Rainfile, Probf, Mvoid)), 1 On routines
locname(Precon1, data(0)), Dofproc2(Midnumber, M dint, Mstrin gs)), 1 Whole
locname(Precon2, data(0)), Dofproc3(Midnumber, M dint, Mstrin gs)), 1 Fixed
locname(Precon3, data(0)), Dofproc3(Rainfile, Mstrin gs, M dch an), 1 Open
locname(Precon4, data(0)), Dofproc4(Midnumber, M dint, M dint, Mstrin gs)), 1 Float
locname(Precon5, data(0)), Dofproc6(Rainfile, Mstrin gs, M dch an, M dint)), 1 Set
locname(Precon6, data(0)), Dofproc6(Rainfile, Mstrin gs, M dch an, M dint, M dint, M dint)), 1 Establish
locname(Codibtv, data(0)), Decodc0(Codibtv)), 1 Sys trace
locname(Codp2ivv, data(0)), Decodc0(Cod2iv, Mvoid)), 1 On tick
locname(Codpibbv, data(0)), Decodc0(Cod1bb, Mvoid)), 1 On error
locname(Codibtv, data(0)), Decodc0(Codibtv, Mvoid)), 1 On sys trace
locname(Codiv, data(0)), Decodc0(M dint, Mvoid)), 1 Warning level

PAGE 1-4  DSKB:SRJABS.BL1(LISONLGA)ECMU-10A  29-Sep-77 22:10  55 blocks

dataarea(Hydi)
  data(0), Locname(Rattine, Delref(Hdtime))
dataarea(Hydi)
  data(0), Gbinam(Hdtime, Mdnmaxpil e)
dataarea(Hydi2)
  data(0), Locname(Codiv, Decodc0(Mvoid)), 1 Warning level

dataarea(Hydi)
  data(0), Locname(Rattine, Delref(Hdtime))
data()
dlocname(Codrtv, 0),
data()
locname(Codrt1, Deftcode0(Raftime, Mdvoid)),
data()
locname(Codrt1, Deftcode0(Mdtime, Mdreal))
dataend;
ZCHK

ZK>XX
dataarea(Kar0)
gbiname(Mdniot, Mdvnumplace),
gbiname(Mntstep, Mdvnumplace)
dataend;
dataarea(Kar1)
data()
locname(Mdrtmask, Deftstruct4(Mdbits, Lexaux, Mbits, Lexgen, Mdbool, Lexorp, Mdbool, Lextmp)),
data()
locname(Rowint, Deftrow(1), 0),
data()
locname(Rowstep, Deftrow(1))
dataend;
dataarea(Kar2)
data()
locname(Mhwait, Deftref(Rowstep))
dataend;
dataarea(Kar3)
data()
locname(Mdpath, Deftstruct2(Mdniot, Lexaux, Mhwait, Lexaux))
dataend;
dataarea(Kar4)
locname(Mdalgparm, Deftunion2(Mdpath, Mdniot)),
locname(Mdpathslot, Deftunion2(Mdpath, Mdniot))
dataend;
dataarea(Kar5)
data()
locname(Mdalglist, Deftrow(Mdalgparm, 1)),
data()
locname(Procstep, Deftproc8(Mntstep)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdvoid)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdbool)),
data()
locname(Procstep, Deftproci(Mdpathslot, Mntstep)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdvoid)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdniot)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdniot)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdnull)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdniot)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdniot)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdvoid)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdniot)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdniot)),
data()
locname(Proclast, Deftproci(Mdpathslot, Mdvoid)),
data()
structure fmode(I, P, S) = (.underneath AadrTincon(I-1))<P, S> ;

map fmode

ZH> X  Reftime: Codp:Codrvt:Codt:Z<ZH
ZH<XX Mdirn: Rouint: Rousten: Mdarglist: Mpath: Mdrtmask:
Prepspvr: Prcpprtv: Prepspv: Prcppsr: Prcpps:
Prcpps: Prpscbl: Prestep: Prepspv:
Code: Codist: Codrvt: Codt: Z<ZH
Rouint: Refile: Refstring:
Rouhool: Rouchar: Rouout: Rouin: Rouout: Rouin:
Mdcomp1: Mdcomp2:
Precint: Precral: Precrr: Precbtrb: Preclibb:
Codibtv: Codp2: Codibtv: Codibtv: Codibtv: Codibtv:

global data(Vector, Idata,
Reftime, Refint[Link], Refile[Link], Refstring[Link],
ZH> X  Reftime[Link], Z<ZH
ZH<XX Mdirn[Link], Z<ZH
9
Rouil, Rouhool[Link], Rouchar[Link], Rouout[Link], Rouin[Link],
Rouout[Link], Rouin[Link],
ZH<XX Rouint[Link], Rousten[Link], Mdarglist[Link], Z<ZH
9
Struct, Mdcomp1[Link], Mdcomp2[Link],
ZH<XX Mdpath[Link], Mdrtmask[Link], Z<ZH

...
bind

INDEXES INTO THE XMODES TABLE (SEE SEMRA.BLI)

Mint = 0, Mlink = 1, Mreal = 2, Mreal = 3, Mcompl = 4, Mcompl = 5,
Mchar = 6, Mstring = 7, Mbool = 8, Mbites = 9,
Mempty = 10, Mvoid = 11.

bind

Obabs = uplift( Hwords(Idmon, Pabs1), Hwords(Mint, Mint),
Hwords(Idmon, Pabs1), Hwords(Mint, Mint),
Hwords(Idmon, Pabs2), Hwords(Mreal, Mreal),
Hwords(Idmon, Pabs2), Hwords(Mreal, Mreal),
Hwords(Idmon, Mbool), Hwords(Mcompl, Mreal),
Hwords(Idmon, Mbool), Hwords(Mcompl, Mreal),
Hwords(Idmon, Mbool), Hwords(Mcompl, Mreal),
Hwords(Idmon, Mbool), Hwords(Mcompl, Mreal),

Oband = uplift( Hwords(Idgen, Pandb), Mbool, Mbool, Mbool,
Hwords(Idgen, Pandb), Mbool, Mbool, Mbool,
Hwords(Idgen, Pandb), Mbool, Mbool, Mbool,

Obarg = uplift( Hwords(Idmon, Parg), Hwords(Mcompl, Mreal),
Hwords(Idmon, Parg), Hwords(Mcompl, Mreal),
Hwords(Idmon, Parg), Hwords(Mcompl, Mreal),

Obbin = uplift( Hwords(Idmon, Pbin), Hwords(Mint, Mbites),

Obcon = uplift( Hwords(Idmon, Pcon), Hwords(Mcompl, Mcompl),
Hwords(Idmon, Pcon), Hwords(Mcompl, Mcompl),

Obdiv = uplift( Hwords(IdRflA, Pdiv),
Hwords(IdRflA, Pdiv),

Obdoum = uplift( Hwords(Idmon, Pdoumsa), Hwords(Mempty, Mvoid),
Hwords(Idmon, Pdoumsa), Hwords(Mempty, Mvoid),

Hwords(Idmon, Pdoumsa), Hwords(Mempty, Mvoid),

Obdvab = uplift Hwords(IdRAR, Pdiv),
Obels = uplift Hwords(Idgen, Pambt), Mbyte, Nblocks,int,
Hwords(Idgen, Pambby), Mchar, Nbytes, int,
Obanti = uplift Hwords(Idmon, Penti), Hwords(Mreal, Mint),
Hwords(Idmon, Penti), Hwords(Mreal, Mint),
Obaq = uplift Hwords(IdANNC, Pqg),
Hwords(IdSSB, Pqgcs),
Hwords(Idgen, Pqgbt), Mbyte, Mblocks, Nblocks,
Hwords(Idgen, Pqgbty), Mbyte, Nbytes, int,
Obyo = uplift Hwords(IdANNC, Pyl),
Hwords(IdSSB, Pylcs),
Hwords(Idgen, Pylbt), Mbyte, Nblocks, Nblocks,
Hwords(Idgen, Pylby), Mbyte, Nbytes, int,
Obgt = uplift Hwords(IdANNC, Pgtl),
Hwords(IdSSB, Pgtcs),
Hwords(Idgen, Pgtby), Mbyte, Nbytes, int,
Oble = uplift Hwords(IdANNC, Ple),
Hwords(IdSSB, Ples),
Hwords(Idgen, Plebt), Mbyte, Nblocks, Nblocks,
Hwords(Idgen, Pleby), Mbyte, Nbytes, int,
Oblang = uplift Hwords(Idmon, Plang), Hwords(Mint, Mint),
Hwords(Idmon, Plangr), Hwords(Mreal, Mint),
Hwords(Idmon, Plangc), Hwords(Mcomp, Mcomp),
Oblevel = uplift Hwords(Idmon, Plenvint), Hwords(Mint, Msema),
Hwords(Idmon, Plenvam), Hwords(Msema, Mint),
Obit = uplift Hwords(IdANNC, Pli),
Hwords(IdSSB, Plics),
Hwords(Idgen, Piliby), Mbyte, Nbytes, int,
Obish = uplift Hwords(IdIR, Pibusb), Mabsent,
Hwords(IdIR, Pibusb), Mabsent,
Hwords(Idmon, Pibusb), Hwords(Mstring, Mint),
Obadsh = uplift Hwords(IdIRI, Pmod),
Obanah = uplift Hwords(IdARA, Psub),
Hwords(IdIRA, Psub),
Obminus = uplift Hwords(IdANA, Pngeo), Hwords(Mint, Mint),
Hwords(Idmon, Pngeo), Hwords(Mint, Mint),
Hwords(Idmon, Pngeo), Hwords(Mreal, Mint),
Hwords(Idmon, Pngeo), Hwords(Mreal, Mint),
Hwords(Idmon, Pngeo), Hwords(Mreal, Mint),
Hwords(Idmon, Pngeo), Hwords(Mreal, Mint),
Hwords(Idmon, Pngeo), Hwords(Mreal, Mint),
Obadod = uplift Hwords(IdIII, Pmod),
Obno = uplift
  Hwords(IdANANC, Pno),
  Hwords(IdSSS, Pnocs),
  Hwords(Idgen, Pnab),  Mbool, Mbool, Mbool,
  Hwords(Idgen, Pnbti), Mbool, Mbits, Mbits,
  Hwords(Idgen, Pnoby), Mbool, Mbytes, Mbytes,
  0 ),

Obnot = uplift
  Hwords(Idmon, Pnotb), Hwords(Mbool, Mbool),
  Hwords(Idmon, Pnotbt), Hwords(Mbits, Mbits),
  0 ),

Obodd = uplift
  Hwords(Idmon, Podd), Hwords(Mint, Mbool),
  Hwords(Idmon, Podd), Hwords(Mint, Mbool),
  0 ),

Obbar = uplift
  Hwords(Idgen, Porbi),  Mbool, Mbool, Mbool,
  Hwords(Idgen, Porbi),  Mbits, Mbits, Mbits,
  0 ),

Obobvab = uplift
  Hwords(IdIAI, Povar),

Obover = uplift
  Hwords(IdIII, Pover),

Obplab = uplift
  Hwords(IdANA, Padd),
  Hwords(IdAN, Padd),
  Hwords(IdSCS, Pcat+4), Mdstring,
  Hwords(IdSCS, Pcat+3), Mdchar,
  0 ),

Obplto = uplift
  Hwords(IdESS, Ppluscols),

Obpltn = uplift
  Hwords(IdIC, Plitln),

Obplus = uplift
  Hwords(IdANA, Padd),
  Hwords(IdSSS, Pcat),
  Hwords(Idmon, Pnosp), Hwords(Mint, Mint),
  Hwords(Idmon, Pnosp), Hwords(Mint, Mint),
  Hwords(Idmon, Pnosp), Hwords(Mreal, Mreal),
  Hwords(Idmon, Pnosp), Hwords(Mreal, Mreal),
  Hwords(Idmon, Pnosp), Hwords(Mcomp, Mcomp),
  Hwords(Idmon, Pnosp), Hwords(Mcomp, Mcomp),
  0 ),

Obopro = uplift
  Hwords(Idmon, Propr), Hwords(Mint, Mchar),

Obrown = uplift
  Hwords(Idmon, Prown), Hwords(Mreal, Mreal),
  Hwords(Idmon, Prown), Hwords(Mreal, Mreal),
  0 ),

Obshi = uplift
  Hwords(Idgen, Pshl),  Mbits, Mdint, Mbits,

Obshr = uplift
  Hwords(Idgen, Pshr),  Mbits, Mdint, Mbits,

Obshrt = uplift
  Hwords(Idmon, Pshrt), Hwords(Mint, Mint),
  Hwords(Idmon, Pshtr), Hwords(Mreal, Mreal),
  Hwords(Idmon, Pshrt), Hwords(Mcomp, Mcomp),
  0 ),

Obsign = uplift
  Hwords(Idmon, Psgni), Hwords(Mint, Mint),

Obtimes = uplift
  Hwords(IdANA, Pmul),
dataarea(Systab)

! Environment enquiries

data( Defstid( 0, M dint, quoted Deftag( 6, 'MAXINT'))) ,
data( Defstid( 1, M real, quoted Deftag( 7, 'MAREAL'))) ,
data( Defstid( 2, M real, quoted Deftag( 8, 'SMALLREAL'))) ,
data( Defstid( 3, M int; quoted Deftag(10, 'MARICHAR'))) ,

! Operations associated with BITS values

data( Defstid( 4, Prevbit, quoted Deftag( 8, 'BITSPACK'))) ,

! Operations associated with BYTES values

data( Defstid( 5, Prevyst, quoted Deftag( 9, 'BYTESPACK'))) ,

! Standard mathematical constants and functions

data( Defstid( 6, M real, quoted Deftag( 2, 'PI'))) ,
data( Defstid( 7, Prevrr, quoted Deftag( 4, 'SQRT'))) ,
data( Defstid( 8, Prevrr, quoted Deftag( 3, 'EXP'))) ,
data( Defstid( 9, Prevrr, quoted Deftag( 2, 'LN'))) ,
data( Defstid(10, Prevrr, quoted Deftag( 3, 'COS'))) ,
data( Defstid(11, Prevrr, quoted Deftag( 6, 'ARCCOS'))) ,
data( Defstid(12, Prevrr, quoted Deftag( 3, 'SIN'))) ,
data( Defstid(13, Prevrr, quoted Deftag( 6, 'ARCSIN'))) ,
data( Defstid(14, Prevrr, quoted Deftag( 3, 'TAN'))) ,
data( Defstid(15, Prevrr, quoted Deftag( 6, 'ARCTAN'))) ,
data( Defstid(16, Prevrr, quoted Deftag(10, 'NEXTRANDOM'))) ,

! Channels

data( Defstid(17, M chn, quoted Deftag(14, 'STANDINCHANNEL'))) ,
data( Defstid(18, M chn, quoted Deftag(15, 'STANDOUTCHANNEL'))) ,
data( Defstid(19, M chn, quoted Deftag(16, 'STANDAIOCHANN')))

! Files and associated operations

data( Defstid(20, Prevch, quoted Deftag( 4, 'CHR'))) ,
data( Defstid(21, Prevfs, quoted Deftag( 8, 'NATEM'))) ,
data( Defstid(22, Prccn, quoted Deftag(18, 'ONLOGICALFILEEND')))
data(Defstid(23, Prcon, quoted Daftag(17, 'ONPHYSICALFILEEND'))),
data(Defstid(24, Prcon, quoted Daftag(9, 'ONPAGEEND'))),
data(Defstid(25, Prcon, quoted Daftag(9, 'ONLINEEND'))),
data(Defstid(26, Prcoast, quoted Daftag(9, 'ESTABLISH'))),
data(Defstid(27, Prclfac, quoted Daftag(4, 'OPEN'))),
data(Defstid(28, Prevvrs, quoted Daftag(9, 'ASSOCIATE'))),
data(Defstid(29, Prevv, quoted Daftag(5, 'CLOSE'))),
data(Defstid(30, Prevv, quoted Daftag(7, 'SCRATCH'))),
data(Defstid(31, Prclif, quoted Daftag(10, 'CHARNUMBER'))),
data(Defstid(32, Prclif, quoted Daftag(10, 'LINENUMBER'))),
data(Defstid(33, Prevv, quoted Daftag(5, 'SPACE'))),
data(Defstid(34, Prevv, quoted Daftag(7, 'HELINE'))),
data(Defstid(35, Prevv, quoted Daftag(7, 'NEWPAGE'))),
data(Defstid(36, Prevv, quoted Daftag(3, 'SET'))),
data(Defstid(37, Prevv, quoted Daftag(5, 'RESET'))),

! Conversion routines

data(Defstid(39, Prce2i, quoted Daftag(5, 'WHOLE'))),
data(Defstid(40, Prce2i, quoted Daftag(5, 'FIXED'))),
data(Defstid(41, Prce2i, quoted Daftag(5, 'FLOAT'))),

! Formatless transport

data(Defstid(42, Prevvout, quoted Daftag(3, 'PUT'))),
data(Defstid(43, Prevvout, quoted Daftag(3, 'GET'))),

! Binary transport

data(Defstid(44, Prevvop, quoted Daftag(6, 'PUTBIN'))),
data(Defstid(45, Prevvop, quoted Daftag(6, 'GETBIN'))),

! Particular prelude

data(Defstid(46, Relfint, quoted Daftag(10, 'LASTRANDNO'))),
data(Defstid(47, Relfint, quoted Daftag(6, 'RANDNO'))),
data(Defstid(48, Relfile, quoted Daftag(7, 'STANDIN'))),
data(Defstid(49, Relfile, quoted Daftag(8, 'STANDOUT'))),
data(Defstid(50, Relfile, quoted Daftag(9, 'STANDBACK'))),
data(Defstid(51, Prevvout, quoted Daftag(5, 'PRINT'))),
data(Defstid(51, Prevvout, quoted Daftag(5, 'WRITE'))),
data(Defstid(52, Prevvout, quoted Daftag(4, 'REMOV'))),
data(Defstid(53, Prevvop, quoted Daftag(6, 'WRITBIN'))),
data(Defstid(54, Prevvop, quoted Daftag(7, 'READBIN'))),
data(Defstid(55, Code2ivv, quoted Daftag(6, 'ONTICK'))),
data(Defstid(56, Code2ivv, quoted Daftag(7, 'ONERROR'))),
data(Defstid(57, Code2ivv, quoted Daftag(10, 'ONSYSTRACE'))),
data(Defstid(58, Code2ivv, quoted Daftag(8, 'SYSTRACE'))),
data(Defstid(59, Codiv, quoted Daftag(12, 'WARRNINGLEVEL'))),

ZHZ

data(Defstid(60, Mchann, quoted Daftag(13, 'CONSCINCHannel'))),
data(Defstid(61, Mchann, quoted Daftag(14, 'CONSCOUTchannel'))),
data(Defstid(62, Mchann, quoted Daftag(16, 'FIXEDPAGEchannel'))),
data(Defstid(63, Mchann, quoted Daftag(14, 'VARPAGEchannel'))),
data(Defstid(64, Codiv, quoted Daftag(7, 'SUBTRACE'))),
data(Defstid(65, Codiv, quoted Daftag(11, 'GETPROCtime'))),
data(Defstid(66, Codiv, quoted Daftag(13, 'STARTPROCtime'))),
data(Defstid(67, Codiv, quoted Daftag(6, 'MUESCS'))),
data(Defstid(68, Mchann, quoted Daftag(14, 'OSFILECHANNEL'))),
Operators

```
data( Dofstopr(0, Obabs, quoted Dofopr(3, 'ABS'))),
data( Dofstopr(0, Obarg, quoted Dofopr(3, 'ARG'))),
data( Dofstopr(0, Obbin, quoted Dofopr(3, 'BIN'))),
data( Dofstopr(0, Obconj, quoted Dofopr(4, 'CONJ'))),
data( Dofstopr(0, Obdown, quoted Dofopr(4, 'DOWN'))),
data( Dofstopr(0, Obequi, quoted Dofopr(5, 'EQUAL'))),
data( Dofstopr(0, Obnot, quoted Dofopr(5, 'NOT'))),
data( Dofstopr(0, Obodd, quoted Dofopr(3, 'ODD'))),
data( Dofstopr(0, Obopr, quoted Dofopr(4, 'REPR'))),
data( Dofstopr(0, Obcoun, quoted Dofopr(5, 'ROUND'))),
data( Dofstopr(0, Obshrl, quoted Dofopr(7, 'SHR'})
```
PROC 1—13 DSI:B:SRTRBS.BLUL ISOA LBMSCPSU—1 OR 29—Sep—77 22:15

blocks

yblnama l Opradab, Ootstopr2l 1, Obmdab)),
distal Dotetop nlt 1, Obovab, quieted Defoprt 6, 'OVER AB'))),
gbinam s( Oprovab , Deli topr2l 1, Obovab)),
distal Oefstopr ll
lip Obplab, quoted Oofoprl 6, 'PIUSAB'))),~~
gblnams l Oprpiab, Oetstopr2l 1, Obplab)),
datal Oofstoprl( 1, Obplto, quieted Detopr( 6, 'PLUSTO'))),
gblnisse( Opreq, Oetsfopr2l.
4, Obaq) ),
distal Def atopnl l 4, Obne, quoted Defoprl 2, 'NE'))),
gblnamol Oprno , Oafetopr2( 4, Obnefl, distal Defstoprlt 5, Obyo, quoted Dofopr( 2,'CT'))),
gblnamol Oprgt , Dofstopr2( 5, Obyt) , gblnnu,e( Oprlt, Dofstopr2l 5, Dbli)),
yblnistie( Oprdiv , Oofstopr2( 7, Obdiv), distal Oofstopr ll 7, Obtiod, quoted Detopr( 3, 'MOD'))) ,
gblnnu,e( Oprtlies, Dofstopr2( 7, Obtices)) ,
yblnistie( Oprdiv , Oofstopr2( 7, Obdiv) ),
distal Oofstopr ll 7, Oboloe, quoted Defoprl 4, 'ELEfl')))1

gblnnu,e( Oprtiaes, Dofstopr2( 7, Obtices)) ,
gbl~inmo( Oprup2, Oofstopr2 l 8, Obup2)) ,
dista l
Oef ,topr l( 8, Oblwb , quieted
Oofepr( 3,'11)6'))),
datal Detetopr ll 8, Dbshi , quoted Oofopr( 3, 'SHI'))),
distal Dotstoprll 8, Obshr, quoted Dotopr( 3, 'SIIR'))),
gblinmuol Oprplbtm , Oofstopr2l 9, Obp iltm ) ) ,
datal —1)
dataend;

! Think about doing IM and RE

undeclare Lexstop, Lexstart, Lextab;
globaldata,(Lexstart,) Lexml(Lxvstart,empty));
globaldata,(Lexstop, ) Lexml(Lxvstop,empty));
globaldata,(Lextab, )

0, Deftag( 0, empty),
0, Deftag( 2, 'IM'),
0, Deftag( 2, 'RE'),
Defbold(Lxvout, 0, 3, 'OUT'),
Defbold(Lxvpar, 0, 3, 'PAR'),
Defbold(Lxvpro, "P", 2, 'PR'),
Defbold(Lxvpro, "P", 7, 'PRNGINT'),
Defbold(Lxvpro, 0, 4, 'PRID'),
Defbold(Lxvpro, 0, 4, 'PROC'),
Defbold(Lxvpro, Mdreal, 4, 'REAL'),
Defbold(Lxvreal, 0, 3, 'REF'),
Defbold(Lxvheap, Gencloc, 6, 'SELOC'),
Defbold(Lxvthdr, Mdema, 4, 'SEMA'),
Defbold(Lxvshort, 0, 5, 'SHORT'),
Defbold(Lxvskip, Mdskip, 4, 'SKIP'),
Defbold(Lxvthdr, Mdstring, 6, 'STRING'),
Defbold(Lxvstruct, 0, 6, 'STRUCT'),
Defbold(Lxvthen, 0, 4, 'THEN'),
Defbold(Lxvto, 0, 2, 'TO'),
Defbold(Lxvboolen, NOtrue, 4, 'TRUE'),
Defbold(Lxvvoid, Mvoid, 4, 'VOID'),
Defbold(Lxvwhile, 0, 5, 'WHILE'),

XH>X
Defbold(Lxvthdr, Mftime, 8, 'PROCTIME'),

XH>X
XK>XX
Defbold(Lxvthdr, Mdpath, 4, 'PATH'),
Defbold(Lxvthdr, Mdatmask, 10, 'RIGHTMASK'),
Defbold(Lxvthdr, Mslot, 4, 'SLOT'),
Defbold(Lxvthdr, Mstep, 4, 'STEP'),
Defbold(Lxvthdr, Mdmask, 4, 'WALK'),

XX<XX
Defpragma(0, 3, 'RES'),
Defpragma(1, 5, 'POINT'),
Defpragma(2, 5, 'UPPER'),
Defpragma(3, 5, 'LOWER'),
Defpragma(4, 8, 'WARNINGS'),
Defpragma(5, 10, 'NOWARNINGS'),
Defpragma(6, 7, 'LISTING'),
Defpragma(7, 9, 'NOListing'),
Defpragma(8, 4, 'PAGE');

end

stadox