```
MODULE Maple;
(*
This is the main module implementing the Maple Machine.  The Maple Machine is a stack
based computer, with MEMORYSIZE bytes of main memory.  Code, data, and the stack
reside in this main memory, resulting in a uniform address range.  The Maple Machine is
specifically geared towards running programs written in the C programming language.  As
a matter of fact, the Machine has been optimized to run one particular C program, namely
the Maple symbolic algebra system from the Symbolic Computation Group at the
University of Waterloo.  The Maple Machine completely hides the architecture, operating
system, and file system of the machine it is running on.  A program running on the Maple
Machine believes it is running on a 32 bit computer, with a linear address space, and words
stored LSB first.  It can use Unix(tm) style system calls, and Unix style directory and file
names.  All conversions are taken care of at the "microcode" level.  Maple Machines are
implemented in whatever reasonable language is available on the target computer.

Author: Stefan M. Vorkoetter (smvorkoetter@watmum.waterloo.edu)
Date: January-February 1989
*)

IMPORT
  Bitmaps, CFileIO, Files, Fonts, Input, Oberon, SYSTEM, TextFrames, TextViewers, Texts, TimeInfo, Viewers;

CONST
  BUFFERLEN = 1024;  (* buffer used for loading Maple.Bin *)
  CARRIAGERETURN = 0DH;  (* carriage return character *)
  CTRLC = 0DX;  (* used to interrupt processing (RETURN key) *)
  HALTCODE = 30;  (* argument of HALT procedure *)
  MAXARGS = 10;  (* maximum number of arguments to an intrinsic function *)
  MEMORYSIZE = 500000;  (* size of virtual machine memory in bytes *)
  NAMELEN = 33;  (* file name length plus one *)
  NEWLINE = 0AH;  (* newline character used by C programs *)
  RETSTACKSIZE = 20000;  (* size of return stack in bytes *)
  SEMICOLON = 59;  (* ASCII code for semicolon *)
  STACKSIZE = 25000;  (* size of evaluation stack in bytes *)
  STDIN = 0;  (* file descriptor of standard input *)
  STDOUT = 1;   (* file descriptor of standard output *)
  STRINGLEN = 2048;  (* maximum size of a string of type stringType *)
  WORDSIZE = 4;  (* size of a machine word in bytes *)

  (* Instruction opcodes *)
  STORE = 1;                BNOT = 48;
  NOTEQUAL = 2;               BOOL = 49;
  ARGADD = 3;                CALL = 50;
  ARG0STORE = 4;              CASESTAT = 51;
  ARG0FETCH = 5;              DEFAULT = 52;
  ARG1STORE = 6;              DUP = 53;
  ARG1FETCH = 7;              GOTO = 54;
  ARG2STORE = 8;              IFSTAT = 55;
  ARG2FETCH = 9;              IFZ = 56;
  FRAMEADD = 10;               NEG = 57;
  FRAME0STORE = 11;             POP = 58;
  FRAME0FETCH = 12;             RETURNSTAT = 59;
  FRAME1STORE = 13;             RETVAL = 60;
  FRAME1FETCH = 14;             ROT = 61;
  FRAME2STORE = 15;             SCALEADD = 62;
  FRAME2FETCH = 16;             STACK = 63;
  FRAME3STORE = 17;             SWAP = 64;
  FRAME3FETCH = 18;             SWITCH = 65;
  FRAME4STORE = 19;             ULESSTHAN = 66;
  FRAME4FETCH = 20;             ULESSOREQ = 67;
  FRAME5STORE = 21;             UGREATERTHAN = 68;
  FRAME5FETCH = 22;             UGREATEROREQ = 69;
```

```
    FRAME6STORE = 23;              FETCHVAL = 70;
    FRAME6FETCH = 24;              WHILEFOR = 71;
    MODULO = 25;                 BITWISEXOR = 72;
    BITWISEAND = 26;               BITWISEOR = 73;
    MULTIPLY = 27;               BITWISENOT = 74;
    ADD = 28;                  ONEBYTE = 75;
    INCREMENT = 29;                TWOBYTE = 76;
    INCRWORD = 30;                 FOURBYTE = 77;
    SUBTRACT = 31;               FUNCALL = 78;
    DECREMENT = 32;                INTRINSIC = 79;
    DECRWORD = 33;                 FASTSWITCH = 80;
    DIVIDE = 34;               SCALEADDFET = 81;
    LESSTHAN = 35;               A0F0FETCH = 82;
    LEFTSHIFT = 36;              F0SCALEADDFET = 83;
    LESSOREQ = 37;               ADDFETCH = 84;
    EQUALTO = 38;                IFEQUAL = 85;
    GREATERTHAN = 39;              WHILEGREATERFOR = 86;
    GREATEROREQ = 40;              FRAMEADDFETCH = 87;
    RIGHTSHIFT = 41;              ROTSTORE = 88;
    FETCH = 42;                STOREPOP = 89;
    MAPCASEID = 43;                ROTSTOREPOP = 90;
    MAPID = 44;                  FUNCALL0 = 91;
    MAPLENGTH = 45;                FUNCALL1 = 92;
    STOREBYTE = 46;               FUNCALL2 = 93;
    FETCHBYTE = 47;               FUNCALL3 = 94;

    (* Intrinsic function codes *)
    EXITUNDER = 1;                 PRINTF = 17;
    ABSFUNC = 2;                  PUTC = 18;
    EXITFUNC = 3;                 SBRK = 19;
    FCLOSE = 4;                 SETJMP = 20;
    FFLUSH = 5;                 SIGNAL = 21;
    FGETC = 6;                 SPRINTF = 22;
    FOPEN = 7;                  STRCAT = 23;
    FPRINTF = 8;                 STRCMP = 24;
    FREAD = 9;                  STRCPY = 25;
    FREOPEN = 10;                  STRLEN = 26;
    PLOT = 11;                STRNCMP = 27;
    FWRITE = 12;                  STRNCPY = 28;
    GETC = 13;                 SYSTEMFUNC = 29;
    GETENV = 14;                  TIME = 30;
    ISATTY = 15;                 TIMES = 31;
    LONGJMP = 16;

TYPE
    nullRecord = RECORD END;
    nullPointer = POINTER TO nullRecord;
    argListType = ARRAY MAXARGS OF LONGINT;
    stringType = ARRAY STRINGLEN OF CHAR;
    bufferType = ARRAY BUFFERLEN OF BYTE;

VAR
    argList: argListType;
    argPtr: LONGINT;
    breakPtr: LONGINT;
    buffer: bufferType;
    ch: CHAR;
    doubleSpace: BOOLEAN;
    envPtr: LONGINT;
    evalStackBasePtr: LONGINT;
    evalStackPtr: LONGINT;
    fp: Files.File;
```

```
  fr: Files.Rider;
  framePtr: LONGINT;
  inputPending: BOOLEAN;
  int1,int2: INTEGER;
  ir: SHORTINT;
  lastInputCode: INTEGER;
  long1,long2,long3: LONGINT;
  parsedSemicolon: BOOLEAN;
  mapleReset: BOOLEAN;
  memory: nullPointer;
  memoryBasePtr: LONGINT;
  numArgs: LONGINT;
  pc: LONGINT;
  reader: Texts.Reader;
  retStack: nullPointer;
  retStackBasePtr: LONGINT;
  retStackPtr: LONGINT;
  retVal: LONGINT;
  short1,short2: SHORTINT;
  sigHandler: LONGINT;
  startTime: LONGINT;
  stdinFrom,stdinTo: LONGINT;
  stdinText: Texts.Text;
  stdoutText,stderrText: Texts.Text;
  stdoutTextFrame,stderrTextFrame: TextFrames.Frame;
  stdoutViewer,stderrViewer: Viewers.Viewer;
  stdoutWriter,stderrWriter: Texts.Writer;
  string1,string2: stringType;
  workString: stringType;

PROCEDURE NewText( name: ARRAY OF CHAR ): Texts.Text;
  VAR
    text: Texts.Text;
  BEGIN
    NEW(text);
    Texts.Open(text,name);
    RETURN text;
  END NewText;

PROCEDURE NewMenu( name: ARRAY OF CHAR; VAR writer: Texts.Writer; log: BOOLEAN ): Texts.Text;
  VAR
    text: Texts.Text;
    font: Fonts.Font;
  BEGIN
    text := NewText("");
    Texts.WriteString(writer,name);
    Texts.WriteLn(writer);
    Texts.WriteLn(writer);
    Texts.WriteString(writer,"System.Close  Edit.Copy  Edit.Grow  Edit.Locate  Edit.Store");
    IF ~log THEN
      Texts.WriteString(writer,"  Maple.Evaluate ~        Press ");
      font := writer.fnt;
      Texts.SetFont(writer,Fonts.This("Syntax10b.Scn.Fnt"));
      Texts.WriteString(writer,"RETURN");
      Texts.SetFont(writer,font);
      Texts.WriteString(writer," to interrupt computation.");
    END;
    Texts.WriteLn(writer);
    TextViewers.Insert(text,0,writer.buf);
    RETURN text;
  END NewMenu;
```

```
PROCEDURE OpenViewer( stdoutFont: ARRAY OF CHAR );
  BEGIN
    IF (stdoutViewer = NIL) OR (stdoutViewer.state = 0) THEN
      Texts.SetFont(stdoutWriter,Fonts.This("Syntax10.Scn.Fnt"));
      stdoutViewer := TextViewers.NewViewer(NewMenu("Maple.Output",stdoutWriter,FALSE),NewText(""),0,Oberon.UX,(
      stdoutTextFrame := stdoutViewer.dsc.next(TextFrames.Frame);
      stdoutText := stdoutTextFrame.text;
    END;
    Texts.SetFont(stdoutWriter,Fonts.This(stdoutFont));
    IF (stderrViewer = NIL) OR (stderrViewer.state = 0) THEN
      stderrViewer := TextViewers.NewViewer(NewMenu("Maple.Log",stderrWriter,TRUE),NewText(""),0,Oberon.SX,Obero
      stderrTextFrame := stderrViewer.dsc.next(TextFrames.Frame);
      stderrText := stderrTextFrame.text;
    END;
  END OpenViewer;

(*
This procedure allocates the memory for the Maple Machine. Space is also allocated for a
subroutine return stack, which does not occupy the main memory space.
*)
PROCEDURE AllocateMemory;
  BEGIN
    SYSTEM.NEW(memory,MEMORYSIZE);
    (* This failure test currently does not work due to a problem with SYSTEM.NEW *)
    IF memory = NIL THEN
      Texts.WriteString(stderrWriter,"maple: could not allocate ");
      Texts.WriteInt(stderrWriter,MEMORYSIZE,1);
      Texts.WriteString(stderrWriter," bytes of memory");
      Texts.WriteLn(stderrWriter);
      TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
      TextFrames.Mark(stdoutTextFrame,1);
      HALT(HALTCODE);
    END;
    memoryBasePtr := SYSTEM.VAL(LONGINT,memory);
    long1 := 0;
    WHILE long1 < MEMORYSIZE DO
      SYSTEM.PUT(memoryBasePtr+long1,0);
      INC(long1);
    END;

    SYSTEM.NEW(retStack,RETSTACKSIZE);
    (* This failure test currently does not work due to a problem with SYSTEM.NEW *)
    IF retStack = NIL THEN
      Texts.WriteString(stderrWriter,"maple: could not allocate ");
      Texts.WriteInt(stderrWriter,RETSTACKSIZE,1);
      Texts.WriteString(stderrWriter," bytes of return stack");
      Texts.WriteLn(stderrWriter);
      TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
      TextFrames.Mark(stdoutTextFrame,1);
      HALT(HALTCODE);
    END;
    retStackBasePtr := SYSTEM.VAL(LONGINT,retStack);
  END AllocateMemory;

(*
This procedure loads the actual executable of the Maple symbolic algebra package. The executable
is expected to be found in the file Maple.Bin. The loading process consists of the actual loading
of the bytes, and the setting of the addresses breakPtr (used by the C sbrk() function),
evalstackBasePtr (address of the base of the stack), and envPtr (used by the C getenv() function).
The stack locations corresponding to argc and argv in the function main() are also initialized to
indicate 1 argument which is simply the null terminated string "Maple". Command line parameters
make very little sense under Oberon.
*)
PROCEDURE LoadBinary;
  BEGIN
    breakPtr := memoryBasePtr;
    fp := Files.Old("Maple.Bin");
    IF fp = NIL THEN
      Texts.WriteString(stderrWriter,"maple: could not read Maple.Bin");
      Texts.WriteLn(stderrWriter);
      TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
      TextFrames.Mark(stdoutTextFrame,1);
      HALT(HALTCODE);
    END;
    Files.Set(fr,fp,0);
    WHILE ~fr.eof DO
      Files.ReadBytes(fr,buffer,BUFFERLEN);
      IF fr.eof THEN
        int1 := BUFFERLEN - fr.res;
      ELSE
        int1 := BUFFERLEN;
      END;
      int2 := 0;
      WHILE int2 < int1 DO
        SYSTEM.PUT(breakPtr,buffer[int2]);
        INC(breakPtr); INC(int2);
      END;
    END;
    Files.Close(fp);
    DEC(breakPtr,WORDSIZE);
    SYSTEM.GET(breakPtr,pc); INC(pc,memoryBasePtr);
    WHILE breakPtr MOD WORDSIZE # 0 DO INC(breakPtr) END;
    evalStackBasePtr := breakPtr;
    INC(breakPtr,STACKSIZE);
    SYSTEM.PUT(evalStackBasePtr,breakPtr-memoryBasePtr); INC(evalStackBasePtr,WORDSIZE);
    SYSTEM.PUT(evalStackBasePtr,LONG(1)); INC(evalStackBasePtr,WORDSIZE);
    SYSTEM.PUT(evalStackBasePtr,evalStackBasePtr-WORDSIZE*2-memoryBasePtr); INC(evalStackBasePtr,WORDSIZE);
    SYSTEM.PUT(breakPtr,SHORT(ORD('M'))); INC(breakPtr);
    SYSTEM.PUT(breakPtr,SHORT(ORD('a'))); INC(breakPtr);
    SYSTEM.PUT(breakPtr,SHORT(ORD('p'))); INC(breakPtr);
    SYSTEM.PUT(breakPtr,SHORT(ORD('l'))); INC(breakPtr);
    SYSTEM.PUT(breakPtr,SHORT(ORD('e'))); INC(breakPtr);
    envPtr := breakPtr;
    SYSTEM.PUT(breakPtr,0); INC(breakPtr);
    WHILE breakPtr MOD WORDSIZE # 0 DO INC(breakPtr) END;
  END LoadBinary;

(*
This function returns the value of the one byte (0 to 127), two byte (-128 to -1), three
byte (-32768 to 32767), or five byte (-2147483648 to 2147483647) constant pointed to
by the program counter, pc.  The program counter is incremented appropriately.
*)
PROCEDURE FetchValue(): LONGINT;
  VAR
    short: SHORTINT;
    int: INTEGER;
    long: LONGINT;
  BEGIN
    SYSTEM.GET(pc,short); INC(pc);
    IF short < 0 THEN
      RETURN LONG(LONG(short)) + 128;
    ELSIF short = ONEBYTE THEN
      SYSTEM.GET(pc,short); INC(pc);
      RETURN LONG(LONG(short));
    ELSIF short = TWOBYTE THEN
      SYSTEM.GET(pc,int); INC(pc,2);
      RETURN LONG(int);
    ELSE
      SYSTEM.GET(pc,long); INC(pc,WORDSIZE);
      RETURN long;
    END;
  END FetchValue;

(*
Here is where characters are read from the Oberon equivalent of the standard input. If there
are more characters in the selected text, return the next one. Otherwise return a semicolon
if one might be needed. Otherwise return a newline if one might be needed. Otherwise
return FALSE as the function value, which will cause the caller to suspend the Maple Machine
until more input is available.
*)
PROCEDURE ExecuteGETC(): BOOLEAN;
  VAR
    c: CHAR;
  BEGIN
    IF (stdinText = NIL) OR (Texts.Pos(reader) >= stdinTo) THEN
      IF ((lastInputCode # NEWLINE) OR (~parsedSemicolon)) & (stdinText # NIL) THEN
        IF parsedSemicolon THEN
          lastInputCode := NEWLINE;
          retVal := NEWLINE;
        ELSE
          lastInputCode := SEMICOLON;
          retVal := SEMICOLON;
          parsedSemicolon := TRUE;
        END;
        RETURN TRUE;
      ELSE
        RETURN FALSE;
      END;
    ELSE
      Texts.Read(reader,c);
      IF c = ';' THEN
        parsedSemicolon := TRUE;
      ELSIF (c >= '!') & (c <= '~') THEN
        parsedSemicolon := FALSE;
      END;
      IF c = CHR(CARRIAGERETURN) THEN
        lastInputCode := NEWLINE;
      ELSE
        lastInputCode := ORD(c);
      END;
      retVal := lastInputCode;
      RETURN TRUE;
    END;
  END ExecuteGETC;

(*
These two routines fetch and store a null terminated string of characters from or to the specified
address, to or from the string parameter. The maximum length of string that can be transferred this
way is STRINGLEN, although this limit is not checked by the code.
*)
PROCEDURE GetString( p: LONGINT; VAR s: stringType );
  VAR
    i: INTEGER;
    short: SHORTINT;
  BEGIN
    i := 0;
```

```
      REPEAT
         SYSTEM.GET(p,short);
         s[i] := CHR(short);
         INC(p); INC(i);
      UNTIL short = 0;
   END GetString;

PROCEDURE PutString( p: LONGINT; VAR s: stringType );
   VAR
      i: INTEGER;
   BEGIN
      i := 0;
      REPEAT
         SYSTEM.PUT(p,SHORT(ORD(s[i])));
         INC(p); INC(i);
      UNTIL s[i-1] = 0X;
   END PutString;
(*
This is the core of the printf(), fprintf(), and sprintf() functions. It is however not a complete
implementation. Only the formats "%s" and "%d" are understood, since these are all that
Maple uses. The last parameter is the index of the first member of argList that is one of the
arguments to be formatted.
*)
PROCEDURE DoPrint( VAR target,format: stringType; first: INTEGER ): LONGINT;
   VAR
      p,q: INTEGER;
      i,j: LONGINT;
   BEGIN
      p := 0; q := 0;
      WHILE format[q] # 0X DO
         WHILE (format[q] # 0X) & (format[q] # '%') DO
            target[p] := format[q];
            INC(p); INC(q);
         END;
         IF format[q] = '%' THEN
            INC(q);
            IF format[q] = 'd' THEN
               IF argList[first] < 0 THEN
                  target[p] := '-';
                  INC(p);
               END;
               i := ABS(argList[first]);
               IF i = 0 THEN
                  target[p] := '0';
                  INC(p);
               ELSE
                  j := 10;
                  WHILE j <= i DO
                     j := j * 10;
                  END;
                  REPEAT
                     j := j DIV 10;
                     target[p] := CHR(i DIV j + ORD('0'));
                     i := i MOD j;
                     INC(p);
                  UNTIL j = 1;
               END;
               INC(first); INC(q);
            ELSIF format[q] = 's' THEN
               GetString(memoryBasePtr+argList[first],workString);
               i := 0;
               WHILE workString[i] # 0X DO
            target[p] := workString[i];
            INC(p); INC(i);
          END;
          INC(first); INC(q);
        ELSIF format[q] # 0X THEN
          target[p] := format[q];
          INC(p); INC(q);
        END;
      END;
    END;
    target[p] := 0X;
    RETURN p;
  END DoPrint;

(*
This procedure is called by the main interpreter whenever a C library function that is actually
implemented in "microcode" is called.  If this procedure returns FALSE, it means that the
function could not be executed further.  This can happen with the fgetc() and getc() functions
if no input is available, or with the exit() and –exit() functions, since they mean "don't
execute any further!".
*)
PROCEDURE ExecuteIntrinsic(): BOOLEAN;
  VAR
    i: INTEGER;
    a,b: LONGINT;
  BEGIN
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr,numArgs);
    WHILE numArgs > 0 DO
      DEC(numArgs);
      DEC(evalStackPtr,WORDSIZE);
      SYSTEM.GET(evalStackPtr,argList[numArgs]);
    END;
    SYSTEM.GET(pc,ir);
    INC(pc);
    CASE ir OF
      FCLOSE:
        retVal := CFileIO.FClose(argList[0]);
    | FFLUSH:
        retVal := CFileIO.FFlush(argList[0]);
    | FGETC,GETC:
        IF argList[0] = STDIN THEN
          IF ~ExecuteGETC() THEN
            inputPending := TRUE;
            RETURN FALSE;
          END;
        ELSE
          retVal := CFileIO.FGetC(argList[0]);
        END;
    | FOPEN:
        GetString(memoryBasePtr+argList[0],string1);
        GetString(memoryBasePtr+argList[1],string2);
        retVal := CFileIO.FOpen(string1,string2);
        IF retVal = 0 THEN
          Texts.WriteString(stderrWriter,"–tried to open ");
        ELSE
          Texts.WriteString(stderrWriter,"–opened ");
        END;
        Texts.WriteString(stderrWriter,string1);
        Texts.WriteLn(stderrWriter);
        TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
    | FPRINTF:
```

```
    GetString(memoryBasePtr+argList[1],string1);
    retVal := DoPrint(string2,string1,2);
    i := 0;
    IF argList[0] = STDOUT THEN
      WHILE string2[i] # 0X DO
        IF string2[i] = CHR(NEWLINE) THEN
          Texts.WriteLn(stdoutWriter);
          IF doubleSpace THEN
            Texts.WriteLn(stdoutWriter);
          END;
          TextViewers.Insert(stdoutText,stdoutText.len,stdoutWriter.buf);
        ELSE
          Texts.Write(stdoutWriter,string2[i]);
        END;
        INC(i);
      END;
      retVal := 0;
    ELSE
      WHILE string2[i] # 0X DO
        retVal := CFileIO.FPutC(string2[i],argList[0]);
        INC(i);
      END;
    END;
| FREAD:
    a := argList[1] * argList[2];
    retVal := 0; b := 1;
    WHILE (a > 0) & (b # 0) DO
      IF a > STRINGLEN THEN
        b := STRINGLEN;
      ELSE
        b := a;
      END;
      b := CFileIO.FRead(string1,1,b,argList[3]);
      i := 0;
      WHILE i < b DO
        SYSTEM.PUT(memoryBasePtr+argList[0]+retVal+i,string1[i]);
        INC(i);
      END;
      retVal := retVal + b;
      a := a - b;
    END;
    retVal := retVal DIV argList[1];
| FREOPEN:
    GetString(memoryBasePtr+argList[0],string1);
    GetString(memoryBasePtr+argList[1],string2);
    retVal := CFileIO.FReopen(string1,string2,argList[2]);
| FWRITE:
    a := argList[1] * argList[2];
    retVal := 0; b := 1;
    WHILE (a > 0) & (b # 0) DO
      IF a > STRINGLEN THEN
        b := STRINGLEN;
      ELSE
        b := a;
      END;
      i := 0;
      WHILE i < b DO
        SYSTEM.GET(memoryBasePtr+argList[0]+retVal+i,string1[i]);
        INC(i);
      END;
      b := CFileIO.FWrite(string1,1,b,argList[3]);
      retVal := retVal + b;
```

```
       a := a - b;
    END;
    retVal := retVal DIV argList[1];
| PLOT:
    (*
    Eventually, I would like to implement the Maple plotting package as an instruction
    in the Maple Machine. This would of course be far easier on a machine where
    the Maple Machine (or at least this part) could be written in C.
    *)
    Texts.WriteString(stderrWriter,"maple: function plot() is not implemented");
    Texts.WriteLn(stderrWriter);
    TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
| PRINTF:
    GetString(memoryBasePtr+argList[0],string1);
    retVal := DoPrint(string2,string1,1);
    i := 0;
    WHILE string2[i] # 0X DO
       IF string2[i] = CHR(NEWLINE) THEN
          Texts.WriteLn(stdoutWriter);
          IF doubleSpace THEN
             Texts.WriteLn(stdoutWriter);
          END;
          TextViewers.Insert(stdoutText,stdoutText.len,stdoutWriter.buf);
       ELSE
          Texts.Write(stdoutWriter,string2[i]);
       END;
       INC(i);
    END;
    retVal := 0;
| PUTC:
    retVal := CFileIO.FPutC(CHR(argList[0]),argList[1]);
| SPRINTF:
    GetString(memoryBasePtr+argList[1],string1);
    retVal := DoPrint(string2,string1,2);
    PutString(memoryBasePtr+argList[0],string2);
    retVal := argList[0];
| EXITFUNC,EXITUNDER:
    mapleReset := FALSE;
    RETURN FALSE;
| LONGJMP:
    SYSTEM.GET(memoryBasePtr+argList[0],pc);
    SYSTEM.GET(memoryBasePtr+argList[0]+WORDSIZE,evalStackPtr);
    SYSTEM.GET(memoryBasePtr+argList[0]+WORDSIZE*2,retStackPtr);
    SYSTEM.GET(memoryBasePtr+argList[0]+WORDSIZE*3,framePtr);
    SYSTEM.GET(memoryBasePtr+argList[0]+WORDSIZE*4,argPtr);
    retVal := argList[1];
| SETJMP:
    SYSTEM.PUT(memoryBasePtr+argList[0],pc);
    SYSTEM.PUT(memoryBasePtr+argList[0]+WORDSIZE,evalStackPtr);
    SYSTEM.PUT(memoryBasePtr+argList[0]+WORDSIZE*2,retStackPtr);
    SYSTEM.PUT(memoryBasePtr+argList[0]+WORDSIZE*3,framePtr);
    SYSTEM.PUT(memoryBasePtr+argList[0]+WORDSIZE*4,argPtr);
    retVal := 0;
| STRCAT:
    long1 := memoryBasePtr + argList[0];
    long2 := memoryBasePtr + argList[1];
    retVal := argList[0];
    SYSTEM.GET(long1,short1);
    WHILE short1 # 0 DO
       INC(long1);
       SYSTEM.GET(long1,short1);
    END;
```

```
      REPEAT
         SYSTEM.GET(long2,short1);
         SYSTEM.PUT(long1,short1);
         INC(long1); INC(long2);
      UNTIL short1 = 0;
  | STRCMP:
      long1 := memoryBasePtr + argList[0];
      long2 := memoryBasePtr + argList[1];
      SYSTEM.GET(long1,short1);
      SYSTEM.GET(long2,short2);
      WHILE (short1 # 0) & (short2 # 0) & (short1 = short2) DO
         INC(long1); INC(long2);
         SYSTEM.GET(long1,short1);
         SYSTEM.GET(long2,short2);
      END;
      IF short1 > short2 THEN
         retVal := 1;
      ELSIF short1 < short2 THEN
         retVal := -1;
      ELSE
         retVal := 0;
      END;
  | STRCPY:
      long1 := memoryBasePtr + argList[0];
      long2 := memoryBasePtr + argList[1];
      retVal := argList[0];
      REPEAT
         SYSTEM.GET(long2,short1);
         SYSTEM.PUT(long1,short1);
         INC(long1); INC(long2);
      UNTIL short1 = 0;
  | STRLEN:
      long1 := memoryBasePtr + argList[0];
      retVal := 0;
      SYSTEM.GET(long1,short1);
      WHILE short1 # 0 DO
         INC(retVal); INC(long1);
         SYSTEM.GET(long1,short1);
      END;
  | STRNCMP:
      long1 := memoryBasePtr + argList[0];
      long2 := memoryBasePtr + argList[1];
      long3 := memoryBasePtr + argList[2];
      SYSTEM.GET(long1,short1);
      SYSTEM.GET(long2,short2);
      WHILE (short1 # 0) & (short2 # 0) & (short1 = short2) & (long3 > 0) DO
         INC(long1); INC(long2);
         SYSTEM.GET(long1,short1);
         SYSTEM.GET(long2,short2);
         DEC(long3);
      END;
      IF long3 = 0 THEN
         retVal := 0;
      ELSIF short1 > short2 THEN
         retVal := 1;
      ELSIF short1 < short2 THEN
         retVal := -1;
      ELSE
         retVal := 0;
      END;
  | STRNCPY:
      long1 := memoryBasePtr + argList[0];
```

```
    long2 := memoryBasePtr + argList[1];
    long3 := memoryBasePtr + argList[2];
    retVal := argList[0];
    IF long3 > 0 THEN
      REPEAT
        SYSTEM.GET(long2,short1);
        SYSTEM.PUT(long1,short1);
        INC(long1); INC(long2);
        DEC(long3);
      UNTIL (short1 = 0) OR (long3 = 0);
      WHILE long3 > 0 DO
        SYSTEM.PUT(long1,0);
        INC(long1); DEC(long3);
      END;
    END;
| SBRK:
    retVal := breakPtr - memoryBasePtr;
    long1 := argList[0];
    WHILE long1 MOD WORDSIZE # 0 DO
      INC(long1);
    END;
    INC(breakPtr,long1);
    IF breakPtr - memoryBasePtr > MEMORYSIZE THEN
      DEC(breakPtr,long1);
      retVal := -1;
    END;
| ABSFUNC:
    retVal := ABS(argList[0]);
| GETENV:
    retVal := envPtr - memoryBasePtr;
| ISATTY:
    IF argList[0] < 3 THEN
      retVal := 1;
    ELSE
      retVal := 0;
    END;
| TIME:
    retVal := TimeInfo.GetTime();
    IF numArgs # 0 THEN
      SYSTEM.PUT(memoryBasePtr+argList[0],retVal);
    END;
| TIMES:
    SYSTEM.PUT(memoryBasePtr+argList[0],(TimeInfo.GetTime()-startTime) * 60);
| SIGNAL:
    retVal := 0;
    IF argList[1] = 1 THEN
      sigHandler := 0;
    ELSE
      sigHandler := argList[1];
    END;
| SYSTEMFUNC:
    Texts.WriteString(stderrWriter,"maple: function system() is not implemented");
    Texts.WriteLn(stderrWriter);
    TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
    retVal := 0;
ELSE
    Texts.WriteString(stderrWriter,"maple: unknown intrinsic function ");
    Texts.WriteInt(stderrWriter,ir,1);
    Texts.WriteLn(stderrWriter);
    TextViewers.Insert(stderrText,stderrText.len,stderrWriter.buf);
    HALT(HALTCODE);
END;
```

```
    RETURN TRUE;
  END ExecuteIntrinsic;

(*
These two routines are called by the main interpreter to execute switch statements. There
are two different switch instructions supported by the Maple Machine. SWITCH is for
switch statements with many widely separated cases, while FASTSWITCH is for switch
statements with closely spaced cases. The instruction to use is decided by the assembler
at assembly/link time.
*)
PROCEDURE ExecuteSwitch;
  BEGIN
    DEC(evalStackPtr,WORDSIZE); SYSTEM.GET(evalStackPtr,long1);
    SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
    LOOP
      SYSTEM.GET(pc,ir);
      IF ir # CASESTAT THEN
        EXIT;
      END;
      INC(pc);
      SYSTEM.GET(pc,long2);
      INC(pc,WORDSIZE);
      IF FetchValue() = long1 THEN
        EXIT;
      END;
      pc := long2 + memoryBasePtr;
    END;
  END ExecuteSwitch;

PROCEDURE ExecuteFastSwitch;
  BEGIN
    DEC(evalStackPtr,WORDSIZE); SYSTEM.GET(evalStackPtr,long1);
    SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
    SYSTEM.GET(pc,long2);
    SYSTEM.GET(pc+WORDSIZE,long3);
    IF (long1 < long2) OR (long1 > long3) THEN
      SYSTEM.GET(pc+WORDSIZE*2,pc); INC(pc,memoryBasePtr);
    ELSE
      SYSTEM.GET(pc+(long1-long2+3)*WORDSIZE,pc); INC(pc,memoryBasePtr);
    END;
  END ExecuteFastSwitch;

(*
This routine, based on code in the Oberon text editor, finds the most recently marked selection
to use as input for Maple. Unlike the text editor, only the text that is within the selected region
is acted upon (the text editor only requires you to select a prefix of the file name to edit).
*)
PROCEDURE FindSelection( VAR text: Texts.Text; VAR begin,end: LONGINT );
  VAR
    frame: Bitmaps.Frame;
    viewer: Viewers.Viewer;
    time: LONGINT;
    x,y,i: INTEGER;
  BEGIN
    text := NIL;
    time := -1;
    x := 0;
    REPEAT
      y := 0;
      REPEAT
        viewer := Viewers.This(x,y);
        frame := viewer.dsc;
        WHILE (frame # NIL) & (frame IS TextFrames.Frame) DO
          WITH frame: TextFrames.Frame DO
            IF (frame.sel > 0) & (frame.time > time) THEN
              text := frame.text;
              begin := frame.selbeg.pos;
              end := frame.selend.pos;
              time := frame.time;
            END;
          END;
          frame := frame.next;
        END;
        INC(y,viewer.H);
      UNTIL y >= Bitmaps.Height;
      INC(x,viewer.W);
    UNTIL x = Bitmaps.Width;
  END FindSelection;
```

(*

This is the main interpreter. The first time it is executed, or any time that the mapleReset **flag has been set to** FALSE, **it loads the** Maple.Bin **file, and starts execution at the** main() **entry point. Between invocations, the variable** startTime **keeps track of the total amount of time that has been spent running Maple since it was loaded or last reset. The flag** inputPending **indicates that Maple was suspended as the result of executing a** getc() **or** fgetc() **function when no input was available, in which case the function is re–executed when Maple is next invoked. My apologies to Professor Gutknecht, who dislikes flag variables. The main loop of the interpreter consists almost entirely of low level** SYSTEM.GET **and** SYSTEM.PUT **statements for maximum speed, a technique that Professor Wirth likes to call "assembly language programming with Oberon syntax". This is necessary because speed is critical here. Every attempt was made to retain safety however.**

*)

```
PROCEDURE Evaluate;
  VAR
    par: Oberon.ParList;
    tf: TextFrames.Frame;
    scanner: Texts.Scanner;
  BEGIN
    par := Oberon.Par();
    tf := par.frame(TextFrames.Frame);
    Texts.OpenScanner(scanner,par.text,par.pos);
    Texts.Scan(scanner);
    doubleSpace := FALSE;
    IF scanner.class = Texts.Name THEN
      int1 := 0;
      WHILE (scanner.s[int1] # 0X) & (scanner.s[int1+1] # 0X) DO
        IF (scanner.s[int1] = '1') & (scanner.s[int1+1] > '4') & (scanner.s[int1+1] <= '9')
        OR (scanner.s[int1] = '2') & (scanner.s[int1+1] >= '0') & (scanner.s[int1+1] <= '9') THEN
          doubleSpace := TRUE;
        END;
        INC(int1);
      END;
      OpenViewer(scanner.s);
    ELSE
      OpenViewer("Gacha12.Scn.Fnt");
    END;
    TextFrames.Mark(stdoutTextFrame,-1);

    IF ~mapleReset THEN
      LoadBinary;
      sigHandler := 0;
      retStackPtr := retStackBasePtr;
      argPtr := evalStackBasePtr - 2 * WORDSIZE;
      framePtr := evalStackBasePtr;
    evalStackPtr := evalStackBasePtr;
    inputPending := FALSE;
    startTime := 0;
    CFileIO.CloseAll;
    mapleReset := TRUE;
  END;
  startTime := TimeInfo.GetTime() - startTime;

  FindSelection(stdinText,stdinFrom,stdinTo);
  IF stdinText # NIL THEN
    Texts.OpenReader(reader,stdinText,stdinFrom);
  END;
  parsedSemicolon := FALSE;
  lastInputCode := 00H;
  IF inputPending THEN
    IF ExecuteGETC() THEN END;
    inputPending := FALSE;
  END;

  LOOP
    SYSTEM.GET(pc,ir);
    INC(pc);
    CASE ir OF
      ADD:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1+long2);
    | SUBTRACT:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1-long2);
    | MULTIPLY:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1*long2);
    | DIVIDE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1 DIV long2);
    | MODULO:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1 MOD long2);
    | LEFTSHIFT:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,ASH(long1,long2));
    | RIGHTSHIFT:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,ASH(long1,-long2));
    | LESSTHAN:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2 < long1 THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | LESSOREQ:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2 <= long1 THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | GREATERTHAN:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2 > long1 THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | GREATEROREQ:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2 >= long1 THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | ULESSTHAN:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2+80000000H < long1+80000000H THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | ULESSOREQ:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2+80000000H <= long1+80000000H THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | UGREATERTHAN:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        IF long2+80000000H > long1+80000000H THEN
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
        ELSE
          SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
        END;
    | UGREATEROREQ:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    IF long2+80000000H >= long1+80000000H THEN
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
    ELSE
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
    END;
| EQUALTO:
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    IF long1 = long2 THEN
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
    ELSE
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
    END;
| NOTEQUAL:
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    IF long1 # long2 THEN
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
    ELSE
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
    END;
| BITWISEAND:
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    SYSTEM.PUT(evalStackPtr-WORDSIZE,
      SYSTEM.VAL(LONGINT,SYSTEM.VAL(SET,long1)*SYSTEM.VAL(SET,long2)));
| BITWISEOR:
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    SYSTEM.PUT(evalStackPtr-WORDSIZE,
      SYSTEM.VAL(LONGINT,SYSTEM.VAL(SET,long1)+SYSTEM.VAL(SET,long2)));
| BITWISEXOR:
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    SYSTEM.PUT(evalStackPtr-WORDSIZE,
      SYSTEM.VAL(LONGINT,SYSTEM.VAL(SET,long1)/SYSTEM.VAL(SET,long2)));
| SCALEADD:
    DEC(evalStackPtr,WORDSIZE);
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    SYSTEM.GET(evalStackPtr,long2);
    SYSTEM.PUT(evalStackPtr-WORDSIZE,long1+long2*WORDSIZE);
| BOOL:
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    IF long1 # 0 THEN
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
    ELSE
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
    END;
| BNOT:
    SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
    IF long1 = 0 THEN
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(1)));
    ELSE
      SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(0)));
    END;
| BITWISENOT:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,-1-long1);
    | INCREMENT:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        INC(long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | DECREMENT:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        DEC(long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | INCRWORD:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        INC(long1,WORDSIZE);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | DECRWORD:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        DEC(long1,WORDSIZE);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | NEG:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,-long1);
    | FETCH:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(memoryBasePtr+long1,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | FETCHBYTE:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(memoryBasePtr+long1,short1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,LONG(LONG(short1)));
    | ADDFETCH:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.GET(memoryBasePtr+long1+long2,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | STORE:
        DEC(evalStackPtr,WORDSIZE*2);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.GET(evalStackPtr+WORDSIZE,long2);
        SYSTEM.PUT(memoryBasePtr+long2,long1);
    | STOREBYTE:
        DEC(evalStackPtr,WORDSIZE*2);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.GET(evalStackPtr+WORDSIZE,long2);
        SYSTEM.PUT(memoryBasePtr+long2,SHORT(SHORT(long1)));
    | MAPLENGTH:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(memoryBasePtr+long1,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,SYSTEM.VAL(LONGINT,SYSTEM.VAL(SET,long1)*SYSTEM.VAL(SET,65535
    | MAPID:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(memoryBasePtr+long1,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,SYSTEM.VAL(LONGINT,SYSTEM.VAL(SET,long1)*SYSTEM.VAL(SET,41287
    | MAPCASEID:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(memoryBasePtr+long1+2,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,SYSTEM.VAL(LONGINT,SYSTEM.VAL(SET,long1)*SYSTEM.VAL(SET,63)));
    | ARGADD:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        INC(long1,argPtr-memoryBasePtr);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | ARG0STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(argPtr,long1);
    | ARG0FETCH:
        SYSTEM.GET(argPtr,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | ARG1STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(argPtr+WORDSIZE,long1);
    | ARG1FETCH:
        SYSTEM.GET(argPtr+WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | ARG2STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(argPtr+WORDSIZE*2,long1);
    | ARG2FETCH:
        SYSTEM.GET(argPtr+WORDSIZE*2,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAMEADD:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        INC(long1,framePtr-memoryBasePtr);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | FRAMEADDFETCH:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(long1+framePtr,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | FRAME0STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr,long1);
    | FRAME0FETCH:
        SYSTEM.GET(framePtr,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAME1STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr+WORDSIZE,long1);
    | FRAME1FETCH:
        SYSTEM.GET(framePtr+WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAME2STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr+2*WORDSIZE,long1);
    | FRAME2FETCH:
        SYSTEM.GET(framePtr+2*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAME3STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr+3*WORDSIZE,long1);
    | FRAME3FETCH:
        SYSTEM.GET(framePtr+3*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAME4STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr+4*WORDSIZE,long1);
    | FRAME4FETCH:
        SYSTEM.GET(framePtr+4*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAME5STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr+5*WORDSIZE,long1);
    | FRAME5FETCH:
        SYSTEM.GET(framePtr+5*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FRAME6STORE:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr,long1);
        SYSTEM.PUT(framePtr+6*WORDSIZE,long1);
    | FRAME6FETCH:
        SYSTEM.GET(framePtr+6*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | A0F0FETCH:
        SYSTEM.GET(argPtr,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(framePtr,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | FETCHVAL:
        SYSTEM.PUT(evalStackPtr,retVal);
        INC(evalStackPtr,WORDSIZE);
    | SCALEADDFET:
        DEC(evalStackPtr,WORDSIZE);
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr,long2);
        SYSTEM.GET(memoryBasePtr+long1+long2*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | F0SCALEADDFET:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(framePtr,long2);
        SYSTEM.GET(memoryBasePtr+long1+long2*WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long1);
    | DUP:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.PUT(evalStackPtr,long1);
        INC(evalStackPtr,WORDSIZE);
    | POP:
        DEC(evalStackPtr,WORDSIZE);
    | STOREPOP:
        DEC(evalStackPtr,WORDSIZE*3);
        SYSTEM.GET(evalStackPtr+WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr+WORDSIZE*2,long2);
        SYSTEM.PUT(memoryBasePtr+long2,long1);
    | SWAP:
        SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
        SYSTEM.GET(evalStackPtr-WORDSIZE*2,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE,long2);
        SYSTEM.PUT(evalStackPtr-WORDSIZE*2,long1);
```

```
  | ROT:
      SYSTEM.GET(evalStackPtr-WORDSIZE*2,long1);
      SYSTEM.PUT(evalStackPtr,long1);
      SYSTEM.GET(evalStackPtr-WORDSIZE,long1);
      SYSTEM.PUT(evalStackPtr-WORDSIZE*2,long1);
      INC(evalStackPtr,WORDSIZE);
  | ROTSTORE:
      SYSTEM.GET(evalStackPtr-WORDSIZE*2,long1);
      SYSTEM.GET(evalStackPtr-WORDSIZE,long2);
      SYSTEM.PUT(evalStackPtr-WORDSIZE*2,long2);
      SYSTEM.PUT(memoryBasePtr+long1,long2);
      DEC(evalStackPtr,WORDSIZE);
  | ROTSTOREPOP:
      SYSTEM.GET(evalStackPtr-WORDSIZE*2,long1);
      SYSTEM.GET(evalStackPtr-WORDSIZE,long2);
      SYSTEM.PUT(memoryBasePtr+long1,long2);
      DEC(evalStackPtr,WORDSIZE*2);
  | STACK:
      DEC(evalStackPtr,WORDSIZE);
      SYSTEM.GET(evalStackPtr,long1);
      INC(evalStackPtr,WORDSIZE*long1);
  | IFSTAT:
      DEC(evalStackPtr,WORDSIZE);
      SYSTEM.GET(evalStackPtr,long1);
      IF long1 # 0 THEN
        INC(pc,WORDSIZE);
      ELSE
        SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
      END;
  | IFZ:
      DEC(evalStackPtr,WORDSIZE);
      SYSTEM.GET(evalStackPtr,long1);
      IF long1 = 0 THEN
        INC(pc,WORDSIZE);
      ELSE
        SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
      END;
  | IFEQUAL:
      DEC(evalStackPtr,WORDSIZE*2);
      SYSTEM.GET(evalStackPtr,long1);
      SYSTEM.GET(evalStackPtr+WORDSIZE,long2);
      IF long1 = long2 THEN
        INC(pc,WORDSIZE);
      ELSE
        SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
      END;
  | WHILEFOR:
      DEC(evalStackPtr,WORDSIZE);
      SYSTEM.GET(evalStackPtr,long1);
      IF long1 = 0 THEN
        INC(pc,WORDSIZE);
      END;
      SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
  | WHILEGREATERFOR:
      DEC(evalStackPtr,WORDSIZE*2);
      SYSTEM.GET(evalStackPtr,long1);
      SYSTEM.GET(evalStackPtr+WORDSIZE,long2);
      IF long2 <= long1 THEN
        INC(pc,WORDSIZE);
      END;
      SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
  | GOTO:
    SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
| SWITCH:
    ExecuteSwitch;
| FASTSWITCH:
    ExecuteFastSwitch;
| CASESTAT:
    INC(pc,WORDSIZE);
    long1 := FetchValue();
| DEFAULT:
    (*
    This instruction is not really necessary, but it makes the assembler much simpler and
    it doesn't cost much to execute, especially since it is rarely encountered.
    *)
    ;
| CALL:
    SYSTEM.PUT(retStackPtr,pc); INC(retStackPtr,WORDSIZE);
    SYSTEM.PUT(retStackPtr,framePtr); INC(retStackPtr,WORDSIZE);
    SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
    DEC(evalStackPtr,WORDSIZE); SYSTEM.GET(evalStackPtr,long1);
    DEC(evalStackPtr,WORDSIZE); SYSTEM.GET(evalStackPtr,pc); INC(pc,memoryBasePtr);
    framePtr := evalStackPtr;
    argPtr := framePtr - long1 * WORDSIZE;
    SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
| FUNCALL:
    SYSTEM.PUT(retStackPtr,pc+WORDSIZE); INC(retStackPtr,WORDSIZE);
    SYSTEM.PUT(retStackPtr,framePtr); INC(retStackPtr,WORDSIZE);
    SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
    SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
    DEC(evalStackPtr,WORDSIZE);
    framePtr := evalStackPtr;
    SYSTEM.GET(framePtr,long1);
    argPtr := framePtr - long1 * WORDSIZE;
    SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
| FUNCALL0..FUNCALL3:
    INC(pc);
    SYSTEM.PUT(retStackPtr,pc+WORDSIZE); INC(retStackPtr,WORDSIZE);
    SYSTEM.PUT(retStackPtr,framePtr); INC(retStackPtr,WORDSIZE);
    SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
    SYSTEM.GET(pc,pc); INC(pc,memoryBasePtr);
    framePtr := evalStackPtr;
    argPtr := framePtr - (ir - FUNCALL0) * WORDSIZE;
    SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
    IF (Input.NofCh() # 0) & (sigHandler # 0) THEN
      Input.Read(ch);
      IF ch = CTRLC THEN
        SYSTEM.PUT(retStackPtr,pc); INC(retStackPtr,WORDSIZE);
        SYSTEM.PUT(retStackPtr,framePtr); INC(retStackPtr,WORDSIZE);
        SYSTEM.PUT(retStackPtr,argPtr); INC(retStackPtr,WORDSIZE);
        SYSTEM.PUT(retStackPtr,evalStackPtr); INC(retStackPtr,WORDSIZE);
        pc := sigHandler + memoryBasePtr;
        argPtr := evalStackPtr;
        framePtr := evalStackPtr;
      END;
    END;
| RETURNSTAT:
    IF retStackPtr = retStackBasePtr THEN
      RETURN;
    END;
    DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,evalStackPtr);
    DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,argPtr);
    DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,framePtr);
    DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,pc);
```

```
      | RETVAL:
          DEC(evalStackPtr,WORDSIZE);
          SYSTEM.GET(evalStackPtr,retVal);
          IF retStackPtr = retStackBasePtr THEN
            RETURN;
          END;
          DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,evalStackPtr);
          DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,argPtr);
          DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,framePtr);
          DEC(retStackPtr,WORDSIZE); SYSTEM.GET(retStackPtr,pc);
      | INTRINSIC:
          IF ~ExecuteIntrinsic() THEN EXIT END;
      | ONEBYTE:
          SYSTEM.GET(pc,short1); INC(pc);
          SYSTEM.PUT(evalStackPtr,LONG(LONG(short1))); INC(evalStackPtr,WORDSIZE);
      | TWOBYTE:
          SYSTEM.GET(pc,int1); INC(pc,2);
          SYSTEM.PUT(evalStackPtr,LONG(int1)); INC(evalStackPtr,WORDSIZE);
      | FOURBYTE:
          SYSTEM.GET(pc,long1); INC(pc,WORDSIZE);
          SYSTEM.PUT(evalStackPtr,long1); INC(evalStackPtr,WORDSIZE);
      ELSE
          SYSTEM.PUT(evalStackPtr,LONG(LONG(ir))+128); INC(evalStackPtr,WORDSIZE);
      END;
    END;
    startTime := TimeInfo.GetTime() - startTime;
    TextFrames.Mark(stdoutTextFrame,1);
  END Evaluate;

PROCEDURE Reset;
  BEGIN
    mapleReset := FALSE;
    Evaluate;
  END Reset;

BEGIN
  Texts.OpenWriter(stdoutWriter);
  stdoutViewer := NIL;
  Texts.OpenWriter(stderrWriter);
  stderrViewer := NIL;
  OpenViewer("Gacha12.Scn.Fnt");
  AllocateMemory;
  mapleReset := FALSE;
END Maple.
```

```
DEFINITION Maple;

PROCEDURE Evaluate;
PROCEDURE Reset;

END Maple.
```

```
MODULE CFileIO;
(*
This module is a collection of procedures that closely emulates the primary file I/O functions of the
C standard I/O library.  It is built on top of the Files module provided by the Oberon system.

Author: Stefan M. Vorkoetter (smvorkoetter@watmum.waterloo.edu)
Date: January-February 1989
*)

IMPORT Files;

CONST
  MAXFILES = 20;
  MINFILE = 3;
  EOF = -1;
  CARRIAGERETURN = 0DH;
  NEWLINE = 0AH;

TYPE
  FileDesc = RECORD
    avail: BOOLEAN;
    read: BOOLEAN;
    binary: BOOLEAN;
    first: BOOLEAN;
    fp: Files.File;
    ride: Files.Rider;
  END;

VAR
  fileTable: ARRAY MAXFILES OF FileDesc;
  i: LONGINT;

(*
This routine converts Unix(tm) style directory/file names into Oberon style file names.  See the comments
in the code below for details of the mapping used.  This routine is called by FOpen and FReopen, so the
conversion is transparent to the application using Unix style names.
*)
PROCEDURE ConvertName( VAR name: ARRAY OF CHAR ): BOOLEAN;
  VAR
    i,j,k: LONGINT;
    isbin: BOOLEAN;
  BEGIN
    (* Change pathnames of the form namex/namey/namez/name.m to namexNameyNamezName.m. *)
    i := 0;
    WHILE name[i] # 0X DO
      IF name[i] = '/' THEN
        j := i;
        REPEAT
          INC(j);
          name[j-1] := name[j];
        UNTIL name[j] = 0X;
        name[i] := CAP(name[i])
      ELSE
        INC(i);
      END;
    END;
    (* File is binary if name ends in .m and text otherwise. *)
    isbin := (i >= 2) & (name[i-2] = '.') & (name[i-1] = 'm');
    (* If length is a little bit more than 32, remove every other character starting with the second. *)
    DEC(i,32);
    j := 1;
    WHILE i > 0 DO
```

```
      k := j;
      REPEAT
        INC(k);
        name[k-1] := name[k];
      UNTIL name[k] = 0X;
      INC(j);
      DEC(i);
    END;
    RETURN isbin;
  END ConvertName;

(*
This procedure takes a file descriptor table index, and opens the specified file using that
table entry.  The field .binary is set to indicate that the file is a binary file.  The file is
assumed to be binary if the name ends in .m, or text otherwise.  This will of course only
be true for Maple, which is what this module was written for.
*)
PROCEDURE OpenFile( VAR name: ARRAY OF CHAR; mode: CHAR; i: LONGINT ): LONGINT;
  VAR
    fi: Files.File;
    ch: CHAR;
  BEGIN
    fileTable[i].binary := ConvertName(name);
    fileTable[i].first := TRUE;
    IF mode = 'r' THEN
      fi := Files.Old(name);
      IF fi = NIL THEN
        RETURN 0;
      ELSE
        fileTable[i].avail := FALSE;
        fileTable[i].read := TRUE;
        fileTable[i].fp := fi;
        Files.Set(fileTable[i].ride, fi, 0);
        IF ~fileTable[i].binary THEN
          Files.Read(fileTable[i].ride, ch);
          IF ch = 0FFX THEN
            (*
            This is a kludge to skip the font information at the beginning of a text file.  This
            only works if the font is Syntax10.Scn.Fnt, or some other font with the same
            number of letters in its name.  The code here should really be more intelligent.
            *)
            Files.Set(fileTable[i].ride, fi, 34);
          ELSE
            Files.Set(fileTable[i].ride, fi, 0);
          END;
        END;
        RETURN i + MINFILE;
      END;
    ELSE
      IF mode = 'w' THEN
        fi := Files.New(name);
        Files.Set(fileTable[i].ride, fi, 0);
      ELSE
        fi := Files.Old(name);
        IF fi = NIL THEN
          RETURN 0;
        ELSE
          Files.Set(fileTable[i].ride, fi, Files.Length(fi));
        END;
      END;
      fileTable[i].avail := FALSE;
      fileTable[i].read := FALSE;
      fileTable[i].fp := fi;
      RETURN i + MINFILE;
    END;
  END OpenFile;

PROCEDURE FOpen( VAR name,mode: ARRAY OF CHAR ): LONGINT;
  BEGIN
    i := 0;
    WHILE (i < MAXFILES) & (~fileTable[i].avail) DO
      INC(i);
    END;
    IF i = MAXFILES THEN
      RETURN 0;
    ELSE
      RETURN OpenFile(name,mode[0],i);
    END;
  END FOpen;

PROCEDURE FClose( f: LONGINT ): LONGINT;
  BEGIN
    DEC(f,MINFILE);
    IF (f >= 0) & (f < MAXFILES) & (~fileTable[f].avail) THEN
      IF fileTable[f].read THEN
        Files.Close(fileTable[f].fp);
      ELSE
        Files.Register(fileTable[f].fp);
      END;
      fileTable[f].avail := TRUE;
      IF fileTable[f].read THEN
        RETURN EOF;
      ELSE
        RETURN 0;
      END;
    END;
    RETURN EOF;
  END FClose;

PROCEDURE CloseAll;
  VAR
    f,g: LONGINT;
  BEGIN
    f := 0;
    WHILE (f < MAXFILES) & (~fileTable[f].avail) DO
      g := FClose(f+MINFILE);
      INC(f);
    END;
  END CloseAll;

PROCEDURE FReopen( VAR name,mode: ARRAY OF CHAR; f: LONGINT ): LONGINT;
  VAR
    fi: Files.File;
    ri: Files.Rider;
  BEGIN
    i := FClose(f);
    DEC(f,MINFILE);
    IF (f >= 0) & (f < MAXFILES) THEN
      RETURN OpenFile(name,mode[0],f);
    ELSE
      RETURN 0;
    END;
  END FReopen;
```

```
PROCEDURE FFlush( f: LONGINT ): LONGINT;
   BEGIN
      DEC(f,MINFILE);
      IF (f >= 0) & (f < MAXFILES) & (~fileTable[f].avail) THEN
         (* Files.Purge(fileTable[f].fp); *)
         IF fileTable[f].read THEN
            RETURN EOF;
         ELSE
            RETURN 0;
         END;
      END;
      RETURN EOF;
   END FFlush;

(*
This routine, and the three below it do the actual I/O.  If the file is a text file, carriage return
character are converted to newline characters (0AX) on input, and newlines are converted
to carriage returns on output.  Thus the application using this package thinks it is following
Unix conventions.
*)
PROCEDURE FRead( VAR buffer: ARRAY OF BYTE; size,nitems,f: LONGINT ): LONGINT;
   VAR
      i: LONGINT;
   BEGIN
      DEC(f,MINFILE);
      IF (f >= 0) & (f < MAXFILES) & (~fileTable[f].avail) & (fileTable[f].read) & (~fileTable[f].ride.eof) THEN
         Files.ReadBytes(fileTable[f].ride, buffer, SHORT(nitems*size));
         IF fileTable[f].ride.eof THEN
            nitems := nitems - fileTable[f].ride.res DIV size;
         END;
         IF ~fileTable[f].binary THEN
            i := 0;
            WHILE i < nitems*size DO
               IF ORD(buffer[i]) = CARRIAGERETURN THEN
                  buffer[i] := NEWLINE;
               END;
               INC(i);
            END;
         END;
         RETURN nitems;
      ELSE
         RETURN 0;
      END;
   END FRead;

PROCEDURE FWrite( VAR buffer: ARRAY OF BYTE; size,nitems,f: LONGINT ): LONGINT;
   VAR
      i: LONGINT;
   BEGIN
      DEC(f,MINFILE);
      IF (f >= 0) & (f < MAXFILES) & (~fileTable[f].avail) & (~fileTable[f].read) THEN
         IF ~fileTable[f].binary THEN
            i := 0;
            WHILE i < nitems*size DO
               IF ORD(buffer[i]) = NEWLINE THEN
                  buffer[i] := CARRIAGERETURN;
               END;
               INC(i);
            END;
         END;
         Files.WriteBytes(fileTable[f].ride, buffer, SHORT(nitems*size));
         RETURN nitems;
```

```
    ELSE
      RETURN 0;
    END;
  END FWrite;

PROCEDURE FGetC( f: LONGINT ): LONGINT;
  VAR
    c: BYTE;
  BEGIN
    DEC(f,MINFILE);
    IF (f >= 0) & (f < MAXFILES) & (~fileTable[f].avail) & (fileTable[f].read) & (~fileTable[f].ride.eof) THEN
      Files.Read(fileTable[f].ride, c);
      IF (~fileTable[f].binary) & (ORD(c) = CARRIAGERETURN) THEN
        c := NEWLINE;
      END;
      IF fileTable[f].ride.eof THEN
        RETURN EOF;
      ELSE
        RETURN ORD(c);
      END;
    ELSE
      RETURN EOF;
    END;
  END FGetC;

PROCEDURE FPutC( c: BYTE; f: LONGINT ): LONGINT;
  BEGIN
    DEC(f,MINFILE);
    IF (f >= 0) & (f < MAXFILES) & (~fileTable[f].avail) & (~fileTable[f].read) THEN
      IF (~fileTable[f].binary) & (ORD(c) = NEWLINE) THEN
        c := CARRIAGERETURN;
      END;
      Files.Write(fileTable[f].ride, c);
      RETURN ORD(c);
    ELSE
      RETURN EOF;
    END;
  END FPutC;

BEGIN
  i := 0;
  WHILE i < MAXFILES DO
    fileTable[i].avail := TRUE;
    INC(i);
  END;
END CFileIO.
```

```
DEFINITION CFileIO;

CONST
   EOF = -1;

PROCEDURE FOpen( VAR name,mode: ARRAY OF CHAR ): LONGINT;
(*
Open file with specified name, where mode is "r" for read, "w" for write, or "a" for append.  Return a small
positive integer if successful, or zero if unsuccessful.
*)

PROCEDURE FClose( f: LONGINT ): LONGINT;
(*
Close file specified by f, returning EOF if file is not an output file, or zero otherwise.
*)

PROCEDURE CloseAll;
(*
Close all files in case some where left open by an earlier session.
*)

PROCEDURE FReopen( VAR name,mode: ARRAY OF CHAR; f: LONGINT ): LONGINT;
(*
Close file specified by f, and then open a new file with the specified name, using f as the file descriptor
number.  Mode is "r" for read, "w" for write, or "a" for append.  Returns f if successful, or zero otherwise.
*)

PROCEDURE FFlush( f: LONGINT ): LONGINT;
(*
Flush any input or output buffers associated with file f.  Return EOF if file is not an output file, or
zero otherwise.
*)

PROCEDURE FRead( VAR buffer: ARRAY OF BYTE; size,nitems,f: LONGINT ): LONGINT;
(*
Read nitems items of size size from file f into buffer of specified size.  Returns actual number of bytes read,
or zero on end of file or error.
*)

PROCEDURE FWrite( VAR buffer: ARRAY OF BYTE; size,nitems,f: LONGINT ): LONGINT;
(*
Write nitems items of size size from buffer of specified size to file f.  Returns actual number of bytes written,
or zero on error.
*)

PROCEDURE FGetC( f: LONGINT ): LONGINT;
(*
Read one character from file f.  Returns the ordinal value of the character if successful, or EOF on
end of file or error.
*)

PROCEDURE FPutC( c: BYTE; f: LONGINT ): LONGINT;
(*
Write one character to file f.  Returns the ordinal value of the character if successful, or EOF on error.
*)

END CFileIO.
```

```
MODULE TimeInfo;
(*
This module is an interface to the real time clock of the Ceres computer. There is currently
only one function, which returns the number of seconds since 80-Mar-01 00:00:00. It
returns the correct number of seconds, taking into account leap years, different lengths
of months, and so on.

Author: Stefan M. Vorkoetter (smvorkoetter@watmum.waterloo.edu)
Date: January-February 1989
Acknowledgements: Based on code written by C. Szyperski.
*)
IMPORT
  SYSTEM;

(*
The constants shown below are for the Ceres-1 computer. When porting to a Ceres-2,
they should be changed to FFFFA000H and FFFFFC00H respectively.
*)
CONST
  CLOCKCHIP = 0FFFC80H;  (* address of clock chip *)
  DUMMY = 0FFFFFCH;  (* dummy read address used for clock chip access *)

VAR
  daysInMonth: ARRAY 12 OF SHORTINT;

PROCEDURE GetTime(): LONGINT;
  VAR
    x,sec,min,hour,day,month,year: SHORTINT;
    numy,numm,numd: LONGINT;

  PROCEDURE ReadReg(no: SHORTINT; VAR val: SHORTINT);
    VAR
      lo, hi, x: SHORTINT;
    BEGIN
      REPEAT
        SYSTEM.PUT(CLOCKCHIP, no); SYSTEM.GET(DUMMY, x);
        SYSTEM.GET(CLOCKCHIP, hi); SYSTEM.GET(DUMMY, x);
        SYSTEM.GET(CLOCKCHIP, lo); SYSTEM.GET(DUMMY, x);
        hi := hi MOD 16; lo := lo MOD 16;
      UNTIL (lo # 15) & (hi # 15);
      val := 10 * hi + lo;
    END ReadReg;

  BEGIN
    SYSTEM.GET(CLOCKCHIP, x); SYSTEM.GET(DUMMY, x);
    SYSTEM.GET(CLOCKCHIP, x); SYSTEM.GET(DUMMY, x);
    REPEAT
      ReadReg(0,sec); ReadReg(1,min); ReadReg(2,hour);
      ReadReg(3,day); ReadReg(4,month); ReadReg(5,year);
      ReadReg(0, x);
    UNTIL sec = x;
    (*
    The following assumptions were made:
      year in [0..99]
      month in [1..12]
      day in [1..31]
      hour in [0..23]
      min in [0..59]
      sec in [0..59]
    The code will have to be changed accordingly for a clock chip that uses a different scheme.
    *)
    (* Calculate number of whole years since 1980-Jan-1 00:00:00 *)
    numy := year - 80;

    (* Adjust for years and months since 1980-Mar-1 00:00:00 *)
    numm := month;
    IF numm < 3 THEN
      INC(numm,9);
      DEC(numy);
    ELSE
      DEC(numm,3);
    END;

    (* Calculate number of days in whole years since 1980-Mar-1 00:00:00 *)
    numd := numy * 365 + numy DIV 4;
    IF numy >= 20 THEN DEC(numd) END; (* 2000 is not a leap year *)

    (* Now add in days for whole months *)
    WHILE numm > 0 DO
      DEC(numm);
      INC(numd,LONG(LONG(daysInMonth[numm])));
    END;

    (* Now add in days in month so far *)
    INC(numd,LONG(LONG(day-1)));

    (* Okay, now we can calculate the number of seconds since 1980-Mar-1 00:00:00 *)
    RETURN ((numd * 24 + hour) * 60 + min) * 60 + sec;
  END GetTime;

BEGIN
  (* Month numbering begins with March = 0 *)
  daysInMonth[0] := 31;
  daysInMonth[1] := 30;
  daysInMonth[2] := 31;
  daysInMonth[3] := 30;
  daysInMonth[4] := 31;
  daysInMonth[5] := 31;
  daysInMonth[6] := 30;
  daysInMonth[7] := 31;
  daysInMonth[8] := 30;
  daysInMonth[9] := 31;
  daysInMonth[10] := 31;
  daysInMonth[11 ] := 28;
END TimeInfo.
```

```
DEFINITION TimeInfo;

PROCEDURE GetTime(): LONGINT;
(*
Returns the exact number of whole seconds since 1980-Mar-01 00:00:00.  This is good
until about 2047 AD when using a signed 32 bit value.
*)

END TimeInfo.
```