11. Input-Output

The semantics of certain I/O facilities will rely on the intuitive notion of "files" ("data sets"), which are collections of data external to the program and organized in a sequential or addressable manner. We shall speak of a "sequential file" or a "direct file" according to the method of organization.

Examples of sequential files are:

- a batch of cards
- a series of printed lines
- input from a keyboard
- data on a tape

An example of a direct file is a collection of data items on a drum, or a disc, with each item identified by a unique index.

The individual logical unit in a file will be called an "image". Each "image" is an ordered sequence of characters.

I/O facilities are introduced through block prefixing. For the purpose of this presentation, this collection of facilities will be described by a class called "BASICIO". The class is not explicitly available in any users program.

The program acts as if it were enclosed in the following block:

```
BASICIO (n) begin
                inspect SYSIN do
                inspect SYSOUT do
                <program>
            end
```

where n is an integer constant representing the length of a printed line as defined for the particular implementation.

Within the definition of the I/O semantics, identifiers in CAPITAL LETTERS represent quantities which are not accessible in a user program. A series of dots is used to indicate that actual coding is either found elsewhere, described informally, or implementation defined.

The overall organization of "BASICIO" is as follows:

```
class BASICIO (LINELENGTH); integer LINELENGTH;
   begin ref (infile) SYSIN;
         ref (infile) procedure sysin;
                      sysin :- SYSIN;
         ref (printfile) SYSOUT;
         ref (printfile) procedure sysout;
                      sysout :- SYSOUT;
         class FILE ..................;
         FILE class infile ...........;
         FILE class outfile ..........;
         FILE class directfile .......;
         outfile class printfile .....;

         SYSIN :- new infile ("SYSIN");
         SYSOUT :- new printfile ("SYSOUT");
         SYSIN.open (blanks(80));
         SYSOUT.open(blanks(LINELENGTH));
         inner;
         SYSIN.close;
         SYSOUT.close;
   end BASICIO;
```

The integer "LINELENGTH" represents the implementation defined number of characters in a printed line.

"SYSIN" and "SYSOUT" represent a card-oriented standard input unit and a printer-oriented standard output unit. A program may refer to the corresponding file objects through "sysin" and "sysout" respectively. Most attributes of these file objects are directly available as a result of the implied connection blocks enclosing the program.

The files "SYSIN" and "SYSOUT" will be opened and closed within "BASICIO", i.e. outside the program itself.

11.1 The class "FILE"

11.1.1 Definition

```
class FILE(NAME,.....); value NAME; text NAME; .....
      virtual:  procedure open, close;
      begin text image;
            Boolean OPEN;
            procedure setpos(i); integer i;
                      image.setpos(i);
            integer procedure pos;
                      pos := image.pos;
            Boolean procedure more;
                      more := image.more;
            integer procedure length;
                      length := image.length;
     ..........
     end FILE;
```

11.1.2 Semantics

Within a program, an object of a subclass of "FILE" is used to represent a file. The following four types are predefined:

"infile" representing a sequential file where input operations (transfer of data from file to program) are available.

"outfile" representing a sequential file where output operations (transfer of data from program to file) are available.

"directfile" representing a direct file with facilities for both input and output.

"printfile" (a subclass of outfile) representing a sequential file with certain facilities oriented towards line printers.

An implementation may restrict, in any way, the use of these classes for prefixing or block prefixing. System defined subclasses may, however, be provided in an implementation.

Each FILE object has a text attribute "NAME". It is assumed that this text value identifies an external file which, through an implementation defined mechanism, remains associated with the FILE object. The effect of several file objects representing the same (external) file is implementation defined.

The variable "image" is used to reference a text value which acts as a "buffer", in the sense that it contains the external file image currently being processed. An implementation may require that "image", at the time of an input or output of an image, refers to a whole text object.

The procedures "setpos", "pos", "more" and "length" are introduced for reasons of convenience.

A file is either "open" or "closed", as indicated by the variable "OPEN". Input or output of images may only take place on an open file. A file is initially closed (except SYSIN and SYSOUT as seen from the program).

The procedures "open" and "close" perform the opening and closing operations on a file. Since the procedures are virtual quantities, they may be redefined completely (i.e. at all access levels) for objects belonging to special purpose subclasses of infile, outfile, etc.

These procedures will be implementation defined, but they must conform to the following pattern.

```
procedure open (T,....); text T; .....
begin if OPEN then ERROR;
      OPEN := true;
      image :- T;
      .........
end open;

procedure close (....); .....
begin
      OPEN := false;
      .............
      image :- notext
end close;
```

The procedures may have additional parameters and additional effects.

11.2 The class "infile"

11.2.1 Definition

```
FILE class infile; virtual: Boolean procedure endfile;
                            procedure inimage;
begin procedure open ...;
                begin ....;
                  ENDFILE := false;
                  image := notext;
                  setpos(length+1)
                end open;
      procedure close .....;
                begin ....;
                  ENDFILE := true
                end;
      Boolean ENDFILE;
      Boolean procedure endfile; endfile := ENDFILE;
      procedure inimage;
                begin
                  if ENDFILE then ERROR;
                  .....;
                  setpos(1)
                end;
      character procedure inchar;
                begin if ¬ more then
                      begin inimage; if ENDFILE then ERROR
                      end;
                      inchar := image.getchar
                end inchar;
      Boolean procedure lastitem;
              begin
              L:    if ENDFILE then lastitem := true else
                    begin
                    M: if ¬ more then
                       begin inimage;
                             go to L;
                       end;
                       if inchar = '_' then go to M else
                                         setpos(pos-1);
                    end;
              end lastitem;
```

```
        integer procedure inint;
                begin text T;
                      if lastitem then ERROR;
                      T :- image.sub(pos,length-pos+1);
                      inint := T.getint;
                      setpos(pos+T.pos-1)
                end inint;
        real procedure inreal; .......;
        integer procedure infrac; .......;
        text procedure intext(w); integer w;
                begin text T; integer m;
                      T :- blanks (w);
                      for m := 1 step 1 until w do
                            T.putchar(inchar);
                      intext :- T;
                end intext;
        .....; ENDFILE := true; ....
end infile;
```

11.2.2. Semantics

An object of the class "infile" is used to represent a sequentially organized input file.

The procedure "inimage" performs the transfer of an external file image into the text "image". A run time error occurs if the text is notext or is too short to contain the external image. If it is longer than the external image, the latter is left adjusted and the remainder of the text is blank filled. The position indicator is set to one.

If an "end of file" is encountered, an implementation defined text value is assigned to the text "image" and the variable "ENDFILE" is given the value true. A call on "inimage" when ENDFILE has the value true is a run time error.

The procedure "open" will give ENDFILE the value false and set "image" to blanks. Otherwise it conforms to the pattern of section 11.1.2.

The procedure "endfile" gives access to the value of the variable ENDFILE.

The remaining procedures provide mechanisms for "item oriented" input, which treat the file as a "continuous" stream of characters with a "position indicator" (pos) which is relative to the first character of the current image.

The procedure "inchar" gives access to and scans past the next character.

If the remainder of the file contains one or more non-blank characters, "lastitem" has the value false, and the position indicator of the file is set to the first non-blank character.

The procedures "inreal" and "infrac" are defined in terms of the corresponding de-editing procedures of "image". Otherwise the definition of either procedure is analogous to that of "inint". These three procedures will scan past and convert a numeric item containing the first non-blank character and contained in one image, excepting an arbitrary number of leading blanks.

The expression "intext(n)" where n is a non-negative integer is a reference to a new text of length n containing the next n characters of the file. "pos" is moved to the following character.

The procedures "inchar" and "intext" may both give access to the contents of the image which corresponds to an "end of file".

Example:

The following piece of program will input a matrix by columns. It is assumed that consecutive elements are separated by blanks or contained in different images. The last element of each column should be followed immediately by an asterisk.

```
begin array a[1:n,1:m] integer i,j;
      procedure error; ......;
      for j := 1 step 1 until m do
      begin for i := 1 step 1 until n-1 do
            begin a[i,j] := inreal;
            if (if sysin.more then inchar ≠ '_' else false)
            then error
            end;
            a[n,j] := inreal;
            if inchar ≠ '*' then error;
next: end;.....;
end
```

11.3 The class "outfile"

11.3.1 Definition

```
FILE class outfile; virtual: procedure outimage;
begin procedure open .......;
                begin ......; setpos(1); end;
      procedure close .....;
                begin ...;
                      if pos ≠ 1 then outimage;
                end close;
      procedure outimage;
                begin if ¬ OPEN then ERROR;
                      .....;
                      image := notext;
                      setpos(1)
                end outimage;
      procedure outchar(c); character c;
                begin if ¬ more then outimage;
                      image.putchar(c)
                end outchar;
```

```
    text procedure FIELD(w); integer w;
       begin if w < 0 ∨ w > length then ERROR;
             if pos + w - 1 > length then outimage;
             FIELD :- image.sub(pos,w);
             setpos(pos+w)
       end FIELD;
    procedure outint(i,w); integer i,w;
       FIELD(w).putint(i);
    procedure outfix(r,n,w); real r; integer n,w;
       FIELD(w).putfix(r,n);
    procedure outreal(r,n,w); real r; integer n,w;
       FIELD(w).putreal(r,n):
    procedure outfrac(i,n,w); integer i,n,w;
       FIELD(w).putfrac(i,n);
    procedure outtext(T); value T; text T;
       FIELD(T.length) := T;
    .....
end  outfile;
```

11.3.2 Semantics

An object of the class "outfile" is used to represent a sequentially organized output file.

The transfer of an image from the text "image" to the file is performed by the procedure "outimage". The procedure will react in an implementation defined way if the image length is not appropriate for the external file. The text is cleared to blanks and the position indicator is set to 1, after the transfer.

The procedure "close" will call "outimage" once if the position indicator is different from 1. Otherwise it conforms to the pattern of section 11.1.2.

The procedure "outchar" treats the file as a "continuous" stream of characters.

The remaining procedures provide facilities for "item-oriented" output. Each item is edited into a subtext of "image", whose first character is the one identified by the position indicator of "image", and of a specified width. The position indicator is advanced by a corresponding amount. If an item would extend beyond the last character of "image", the procedure "outimage" is called implicitly prior to the editing operation.

The procedures "outint", "outfix", "outreal" and "outfrac" are defined in terms of the corresponding editing procedures of "image". They have an additional <u>integer</u> parameter which specifies the width of the subtext into which the item will be edited.

For the procedure "outtext", the item width is equal to the length of the text parameter. Notice that this parameter is called by value, which means that a text value is an acceptable actual parameter of "outtext".

11.4 The class "directfile"

Note: The definition of "directfile" is presently under study by a Technical Committee appointed by the SIMULA Standards Group.

11.4.1 Definition

```
FILE class directfile; virtual: Boolean procedure endfile;
                                procedure locate,inimage,outimage;
begin integer LOC;
      integer procedure location; location := LOC;
      procedure locate(i); integer i;
         begin if ¬OPEN then ERROR;
                  .....;
                  LOC := i
         end locate;
      procedure open .....;
         begin
                  .......;
                  setpos(1);
                  locate(1);
         end open;
      procedure close .....;
      Boolean procedure endfile; .....;
      procedure inimage;
         begin .....;
                  locate (LOC+1);
                  setpos(1)
         end inimage;
      procedure outimage;
         begin .....;
                  locate (LOC+1);
                  image := notext;
                  setpos(1)
         end outimage;
```

```
        character procedure inchar .....;
        Boolean procedure lastitem .....;
        integer procedure inint ........;
        real procedure inreal ..........;
        integer procedure infrac .......;
        text procedure intext ..........;
        procedure outchar ..............;
        text procedure FIELD ...........;
        procedure outint ...............;
        procedure outfix ...............;
        procedure outreal ..............;
        procedure outfrac ..............;
        procedure outtext ..............;
        .........
    end directfile;
```

11.4.2 Semantics

An object of the class "directfile" is used to represent an external file in which the individual images are addressable by ordinal numbers.

The variable "LOC" normally contains the ordinal number of an external image. The procedure "location" gives access to the current value of LOC. The procedure "locate" may be used to assign a given value to the variable. The assignment may be accompanied by implementation defined checks and possibly by instructions to an external memory device associated with the given file.

The procedure "open" will locate the first image of the file. Otherwise it conforms to the rules of section 11.1.2.

The procedure "endfile" may have the value true only if the current value of LOC does not identify an image of the external file. The procedure is implementation defined.

The procedure "inimage" will transfer into the text "image" a copy of the external image currently identified by the variable LOC, if there is one. Then the value of LOC is increased by one through a "locate" statement. If the file does not contain an image with an ordinal number equal to the value of LOC, the effect of the procedure "inimage" is implementation defined. The procedure is otherwise analogous to that of section 11.2.

The procedure "outimage" will transfer a copy of the text value "image" to the external file, thereby adding to the file an external image whose ordinal number is equal to the current value of LOC. A run time error occurs if the file cannot be made to contain the image. If the file contains another image with the same ordinal number, that image is deleted. The value of LOC is then increased by one through a "locate" statement. The procedure "outimage" is otherwise analogous to that of section 11.3.

The remaining procedures are analogous to the corresponding procedures of section 11.2 and 11.3.

11.5 The class "printfile"

11.5.1 Definition

```
outfile class printfile;
begin integer LINES PER PAGE, SPACING, LINE;
      integer procedure line; line := LINE;
      procedure lines per page (n); integer n;
                LINES PER PAGE := n;
      procedure spacing(n); integer n;
                SPACING := n;
```

```
        procedure eject(n); integer n;
           begin if ¬ OPEN then ERROR;
                 if n > LINES PER PAGE then n := 1;
                 if LINE > LINES then LINE := 0;
                 ...;
                 LINE := n;
           end eject;
        procedure open ... ;
           begin ..... ; setpos(1); eject(1)end
        procedure close ... ;
           begin ... ;
                 if pos ≠ 1 then outimage;
                 SPACING := 1;
                 eject (LINES PER PAGE);
                 LINES PER PAGE := ... ;
                 LINE := 0
           end;
        procedure outimage;
           begin if ¬ OPEN ∨ image == notext then ERROR;
                 if LINE > LINES PER PAGE then eject (1);
                 comment output the image on the line
                         denoted by LINE;
                 LINE := LINE + SPACING;
                 image : = notext;
                 setpos (1);
           end;
        LINES PER PAGE := ... ;
        SPACING := 1;
end printfile;
```

11.5.2 Semantics

An object of the class "printfile" is used to represent a printer-oriented output file. The class is a subclass of "outfile". A file image represents a line on the printed page.

The variable "LINES PER PAGE" indicates the maximum number of physical lines that will be printed on each page, including intervening blank lines. An implementation defined value is assigned to the variable at the time of object generation, and when the printfile is closed. The procedure "lines per page" may be used to change the value. If the parameter to "lines per page" is zero, "LINES PER PAGE" is reset to the same implementation defined value as at the time of object generation. The effect is implementation defined if the parameter is less than zero.

The variable "SPACING" represents the value by which the variable "LINE" will be incremented after the next printing operation. The variable is set equal to 1 at the time of object generation and when the printfile is closed. Its value may be changed by the procedure "spacing". A call on the procedure "spacing" with a parameter less than zero or greater than "LINES PER PAGE" constitutes an error. The effect of a parameter to "spacing" which is equal to zero may be defined by an implementation either to mean successive printing operations on the same physical line, or to be an error.

The variable "LINE" indicates the ordinal number of the next line to be printed, provided that no implicit or explicit "eject" statement occurs. Its value is accessible through the procedure "line". Note that the value of "LINE" may be greater than "LINES PER PAGE". The value of "LINE" is zero when the file is not open.

The procedure "eject is used to position to a certain line identified by the parameter, n.

The following cases can be distinguished:

n $\leq$0: ERROR
n >LINES PER PAGE: Equivalent to eject (1)
n $\leq$LINE: Position to line number n on the next page
n >LINE: Position to line number n on the current page.

The tests above are performed in the given sequence.

The procedure "outimage" operates according to the rules of section 11.3. In addition, it will update the variable "LINE".

The procedure "open" and "close" conform to the rules of section 11.1. In addition, "open" will position to the top of a page, and "close" will output the current value of "image" if "pos" is different from one and reset "LINE", "SPACING" and "LINES PER PAGE".