```
-- StreamIO.Mesa  Edited by Sandman on July 22, 1977  11:07 AM

DIRECTORY
  IODefs: FROM "iodefs",
  InlineDefs: FROM "inlinedefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs";

DEFINITIONS FROM StreamDefs, IODefs;

StreamIO: PROGRAM [Input, Output: StreamHandle]
  IMPORTS StreamDefs, StringDefs EXPORTS IODefs SHARES IODefs, StreamDefs = PUBLIC

BEGIN

BeginLine: PRIVATE BOOLEAN ← TRUE;

GetInputStream: PROCEDURE RETURNS [StreamHandle] =
  BEGIN
  RETURN[Input]
  END;

GetOutputStream: PROCEDURE RETURNS [StreamHandle] =
  BEGIN
  RETURN[Output]
  END;


-- Character operations

ReadChar: PROCEDURE RETURNS [CHARACTER] =
  BEGIN
  RETURN[Input.get[Input]];
  END;


WriteChar: PROCEDURE [c:CHARACTER] =
  BEGIN
  Output.put[Output, c];
  BeginLine ← c = CR;
  END;


-- Reading Strings

ReadString: PROCEDURE [s:STRING, t:PROCEDURE[CHARACTER]RETURNS[BOOLEAN]] =
  BEGIN
  WriteChar[ReadEditedString[s,t,TRUE]];
  END;

ReadID: PROCEDURE [s:STRING] =
  BEGIN
  [] ← ReadEditedString[s,atomfound,TRUE];
  END;

ReadLine: PROCEDURE [s:STRING] =
  BEGIN
  [] ← ReadEditedString[s,crfound,TRUE];
  WriteChar[CR];
  END;

crfound: PRIVATE PROCEDURE [c:CHARACTER] RETURNS [BOOLEAN] =
  BEGIN RETURN [c = CR] END;

atomfound: PRIVATE PROCEDURE [c:CHARACTER] RETURNS [BOOLEAN] =
  BEGIN RETURN [c = CR OR c = SP] END;

Rubout: SIGNAL = CODE;
LineOverflow: SIGNAL [s: STRING] RETURNS [ns: STRING] = CODE;

-- the editing characters
controlA: CHARACTER = 1C;     -- delete character
controlH: CHARACTER = 10C;    -- delete character
controlW: CHARACTER = 27C;    -- delete word
controlQ: CHARACTER = 21C;    -- delete word
controlX: CHARACTER = 30C;    -- delete line
```

```
controlR: CHARACTER = 22C;    -- retype line
controlV: CHARACTER = 26C;    -- quote next character
ESC: CHARACTER = 33C;         -- use old string

ReadEditedString: PROCEDURE [s:STRING, t:PROCEDURE [CHARACTER] RETURNS [BOOLEAN], newstring:BOOLEAN] RE
**TURNS[CHARACTER] =
  BEGIN
  c: CHARACTER;
  i: CARDINAL;
  state: {TrailingInvisible, Visible, LeadingInvisible};
  c ← Input.get[Input];
  IF newstring THEN
    IF c = ESC THEN
      BEGIN  WriteString[s];  c ← Input.get[Input];  END
    ELSE s.length ← 0;
  UNTIL t[c] DO SELECT c FROM
    DEL => SIGNAL Rubout;
    controlA, controlH =>
      BEGIN
      IF s.length > 0 THEN
        BEGIN
        WITH Output SELECT FROM
          Display => ClearDisplayChar[Output,s[s.length-1]];
          ENDCASE => Output.put[Output,c];
        s.length ← s.length-1;
        END;
      END;
    controlW, controlQ =>
      BEGIN  -- text to be backed up is of the form
      -- ...<LeadingInvisible><Visible><TrailingInvisible>
      -- the <Visible> and <TrailingInvisible> are to be removed.
      state ← TrailingInvisible;
      FOR i DECREASING IN [0..s.length) DO
        SELECT s[i] FROM
          IN ['A..'Z], IN ['a..'z], IN ['0..'9] =>
            IF state = TrailingInvisible THEN state ← Visible;
          ENDCASE =>
            IF state = Visible THEN state ← LeadingInvisible;
        IF state = LeadingInvisible THEN GO TO Done;
        WITH Output SELECT FROM
          Display => ClearDisplayChar[Output,s[i]];
          ENDCASE => Output.put[Output,c];
        REPEAT
          Done => s.length ← i+1;
          FINISHED => s.length ← 0;
        ENDLOOP;
      END;
    controlR =>
      BEGIN
      WriteChar[CR];
      WriteString[s];
      END;
    controlX =>
      BEGIN
      WITH Output SELECT FROM
        Display => ClearCurrentLine[Output];
        ENDCASE => Output.put[Output,c];
      s.length ← 0;
      END;
    controlV =>
      BEGIN
      WHILE s.length >= s.maxlength DO
        s ← SIGNAL LineOverflow[s];
        ENDLOOP;
      s[s.length] ← c ← Input.get[Input];
      s.length ← s.length+1;
      WriteChar[c];
      END;
    ENDCASE =>
      BEGIN
      WHILE s.length >= s.maxlength DO
        s ← SIGNAL LineOverflow[s];
        ENDLOOP;
      s[s.length] ← c;
      s.length ← s.length+1;
      WriteChar[c];
```

```
      END;
    c ← Input.get[Input];
    ENDLOOP;
  RETURN[c];
  END;


-- Writing Strings

WriteString: PROCEDURE [s:STRING] =
  BEGIN
  i:CARDINAL;
  FOR i IN [0..s.length) DO
    Output.put[Output,s[i]];
    ENDLOOP;
  IF s.length # 0 THEN BeginLine ← s[s.length-1] = CR;
  END;

WriteLine: PROCEDURE [s:STRING] =
  BEGIN
  WriteString[s];
  WriteChar[CR];
  END;

NewLine: PROCEDURE RETURNS[BOOLEAN] =
  BEGIN RETURN[BeginLine] END;
```

```
  --  Numerical i/o

  MaxInteger: INTEGER = 32767;   -- maximum positive 16-bit integer
  MaxDigits: PRIVATE INTEGER = 5;   -- ceiling[log10[MaxInteger]]
  DigitIndex: PRIVATE TYPE = [0 .. MaxDigits);


  ReadNumber: PROCEDURE [default: UNSPECIFIED, radix: CARDINAL]
    RETURNS [UNSPECIFIED] =
    BEGIN OPEN InlineDefs;
    CharZero: CARDINAL = LOOPHOLE['0];
    s: STRING ← [10];
    c: ARRAY [0..6) OF [0..9];
    cp, i: CARDINAL ← 0;
    IF radix = 10 AND default < 0 THEN
      BEGIN default ← -default; s[0] ← '-; cp ← 1 END;
    DO
      [default,c[i]] ← LDIVMOD[default,0,radix];
      IF default = 0 THEN EXIT;
      i ← i + 1;
      ENDLOOP;
    FOR i DECREASING IN [0..i] DO
      s[cp] ← LOOPHOLE[c[i] + CharZero,CHARACTER];
      cp ← cp + 1;
     ENDLOOP;
    IF radix = 8 THEN
      BEGIN s[cp] ← 'B; cp ← cp + 1 END;
    s.length ← cp;
    [] ← ReadEditedString[s, atomfound, TRUE];
    RETURN[StringDefs.StringToNumber[s,radix]];
    END;

  ReadDecimal: PROCEDURE RETURNS [INTEGER] =
    BEGIN  -- reads a decimal number in [-MaxInteger .. MaxInteger]
    s: STRING ← [MaxDigits+1];
    [] ← ReadEditedString[s, atomfound, TRUE];
    RETURN [StringDefs.StringToDecimal[s]]
    END;

  ReadOctal: PROCEDURE RETURNS [UNSPECIFIED]=
    BEGIN  -- reads an octal number in [0 .. 177777B]
    s: STRING ← [7];
    [] ← ReadEditedString[s, atomfound, TRUE];
    RETURN [StringDefs.StringToOctal[s]]
    END;

  OutNumber: PROCEDURE
    [stream: StreamHandle, val: INTEGER, format: NumberFormat] =
    BEGIN
    n, b, rd, t: INTEGER;
    fill: CHARACTER;
    neg: BOOLEAN;
    xn: PROCEDURE =
      BEGIN OPEN InlineDefs;
      CharZero: CARDINAL = LOOPHOLE['0];
      CharTen: CARDINAL = LOOPHOLE['A, CARDINAL] - 10;
      r: INTEGER;
      IF n=0 THEN
        BEGIN
        THROUGH (t..rd] DO stream.put[stream,fill] ENDLOOP;
        IF neg THEN stream.put[stream,'-];
        END
      ELSE
        BEGIN
        [n,r] ← LDIVMOD[n,0,b];
        t←t+1;
        xn[];
        stream.put[stream, r + (IF r>9 THEN CharTen ELSE CharZero)];
        END;
      END;

    BEGIN OPEN format;
    t←0; neg←FALSE;
    fill←(IF zerofill THEN '0 ELSE ' );
```

```
  IF val<0 AND ~unsigned THEN
    BEGIN
    val←-val;
    t←1;
    neg←TRUE;
    END;

  b←base;
  rd←columns;
  IF neg AND zerofill THEN BEGIN stream.put[stream,'-]; neg←FALSE END;

  IF (n←val) = 0 THEN t←t+1;     -- 0 is a special case
  xn[];
  IF val = 0 THEN stream.put[stream,'0];
  IF stream = Output THEN BeginLine ← FALSE;
  END;       -- of OPEN block
  END;

WriteNumber: PROCEDURE [v: UNSPECIFIED, f: NumberFormat] =
  BEGIN
  OutNumber[Output,v,f];
  END;

WriteDecimal: PROCEDURE [n: INTEGER] =
  BEGIN
  WriteNumber[n,NumberFormat[10,FALSE,FALSE,0]]
  END;

WriteOctal: PROCEDURE [n: UNSPECIFIED] =
  BEGIN
  WriteNumber[n,NumberFormat[8,FALSE,TRUE,0]];
  IF n ~IN[0..7] THEN WriteChar['B]
  END;

 -- the main body

 IF Input = NIL THEN Input ← StreamDefs.GetDefaultKey[];
 IF Output = NIL THEN Output ← StreamDefs.GetDefaultDisplayStream[];

END.
```