```
Thu Jan 23 19:08:14 EST 1997

************************
PRINT.ME.FOR.LISTINGS.ps   -- Use lpr to get a listing of the *.tu
                              *.dem, *.in and *.out files
Save paper      It is a postscript file
************************

README      - this file

a2spec.ps   - Assignment 2 specification (postcript)
cover2.ps   - Assignment 2 cover sheet

debug.tu    - Debug module
matelem.tu  - MatrixElement class implementation
sample.in   - Sample input for demonstration
sample.out  - Sample output created from sample.in using spmat.dem
spmat.dem   - Interactive demonstration program
spmat.tu    - Part 1 of the SparseMatrix class implementation
spmat2.tu   - Part 2 of the SparseMatrix class implementation
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File:   debug.tu.tu
% Author: Gunnar Gotshalks
% Date:   1997 January 21
%
% This file implements a set of flags useful for writing debugging statements
% that can be executed or not depending upon the value of a flag.
%
% The flag variables are exported as read only for efficiency and ease of
% use.  Operations Set, Reset and Toggle are provided to change the value of
% a flag.  The use of an integer parameter makes it easier to use in
% demonstration programs.

unit module Debug
    export Initialize, Finalize, Set, Reset, Toggle,
           NumberOfFlags, flag0, flag1, flag2, flag3, flag4,
           flag5, flag6, flag7, flag8, flag9

const NumberOfFlags := 10


var flag0, flag1, flag2, flag3, flag4,
    flag5, flag6, flag7, flag8, flag9  : boolean

% Initial conditions have all flags false.

procedure Initialize()
    flag0 := false ; flag1 := false ; flag2 := false
    flag3 := false ; flag4 := false ; flag5 := false
    flag6 := false ; flag7 := false ; flag8 := false
    flag9 := false
end Initialize

procedure Finalize()
    % Nothing to do for this implementation
end Finalize

procedure Set(flagNumber : int)
pre 0 <= flagNumber and flagNumber <= 9
    case flagNumber of
        label 0 : flag0 := true
        label 1 : flag1 := true
        label 2 : flag2 := true
        label 3 : flag3 := true
        label 4 : flag4 := true
        label 5 : flag5 := true
        label 6 : flag6 := true
        label 7 : flag7 := true
        label 8 : flag8 := true
        label 9 : flag9 := true
    end case
end Set

procedure Reset(flagNumber : int)
pre 0 <= flagNumber and flagNumber <= 9
    case flagNumber of
        label 0 : flag0 := false
        label 1 : flag1 := false
        label 2 : flag2 := false
        label 3 : flag3 := false
        label 4 : flag4 := false
        label 5 : flag5 := false
        label 6 : flag6 := false
        label 7 : flag7 := false
        label 8 : flag8 := false
        label 9 : flag9 := false
    end case
end Reset

procedure Toggle(flagNumber : int)
pre 0 <= flagNumber and flagNumber <= 9
    case flagNumber of
        label 0 : flag0 := ~flag0
        label 1 : flag1 := ~flag1
        label 2 : flag2 := ~flag2
        label 3 : flag3 := ~flag3
        label 4 : flag4 := ~flag4
        label 5 : flag5 := ~flag5
        label 6 : flag6 := ~flag6
        label 7 : flag7 := ~flag7
        label 8 : flag8 := ~flag8
        label 9 : flag9 := ~flag9
    end case
end Toggle

end Debug
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File:   matelem.tu
% Author: Gunnar Gotshalks
% Date:   1997 January 21
%
% This file implements the element class for sparse matrices.
%
% In a sparse matrix each element not only has a data value but also records
% its row and column position, and pointers to the next row and column
% elements.

unit class MatrixElement
    export Initialize, Finalize,
           GetData, GetCol, GetRow, GetNextInRow, GetNextInCol,
           SetData, SetCol, SetRow, SetNextInRow, SetNextInCol

type DataType : int

var data      : DataType       % The data for the matrix element.
var row       : int            % The row for the matrix element.
var col       : int            % The col for the matrix element.
var nextInRow : ^MatrixElement % Point to the next element in the row.
var nextInCol : ^MatrixElement % Point to the next element in the column.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Constructor and destructor operations.
%
% Initialize  must be the first procedure called when using a matrix element.
%
% Precondition:  True
% Postcondition: The matrix element has row, column and data set.  Pointers
%       are nil.

procedure Initialize(theRow, theCol : int, theData : DataType)
    row  := theRow
    col  := theCol
    data := theData
    nextInRow := nil
    nextInCol := nil
end Initialize

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Finalize must be the last procedure called when using a matrix element.
%
% Precondition:  Matrix element has been initialized.
% Postcondition: All heap objects pointed to by the matrix element
%       are finalized and freed.

procedure Finalize()
    % Nothing to do for this implementation.
end Finalize

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Accessor operations
%    Return data from a matrix element with no side effects.
%
% Precondition all: The corresponding matrix element fields have been
%       initialized by either using Initialize or the corresonding set
%       operation.
% Postcondition all: Trivially implied by routine name.

function GetData() : DataType
    result data
end GetData

function GetRow() : int
    result row
end GetRow

function GetCol() : int
    result col
end GetCol

function GetNextInRow() : ^MatrixElement
    result nextInRow
end GetNextInRow

function GetNextInCol(): ^MatrixElement
    result nextInCol
end GetNextInCol

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set operations
%    Write data to fields in a matrix element with no other side effects.
%
% Precondition all: True.
% Postcondition all: Trivially implied by routine name.

procedure SetData(theData : DataType)
    data := theData
end SetData

procedure SetRow(theRow : int)
    row  := theRow
end SetRow

procedure SetCol(theCol : int)
    col  := theCol
end SetCol

procedure SetNextInRow(theElem : ^MatrixElement)
    nextInRow := theElem
end SetNextInRow

procedure SetNextInCol(theElem : ^MatrixElement)
    nextInCol := theElem
end SetNextInCol

end MatrixElement
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File:   spmat.tu
% Author: Gunnar Gotshalks
% Date:   1997 January 21
%
% This file contains Part 1/2 the sparse matrix class.  Part 2/2 is in the
% file spmat2.tu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Nonzero elements in a sparse matrix are stored simultaneously in row and
% column lists.  Zero elements are not explicitly stored but are represented
% by their absence.  There is one list for each column and one list for each
% row.  There is a maximum number, MaxRowsCols, for this implementation of
% a sparse matrix because the headers for the column lists, colList, and the
% headers for the row lists, rowList, are fixed size vectors.

unit class SparseMatrix
    import MatrixElement in "matelem.tu",
           Debug in "debug.tu"

    export Initialize, Finalize,
           GetCount, GetMaxRow, GetMaxCol, GetMaxRowsCols,
           GetCountByRow, GetCountByCol, PutElemByRow, PutElemByCol,
           InitIterationByRow, NextIterRowElem,
           Insert, Put, Sum, GetData, ExistsElement

const MaxRowsCols := 100  % Maximum number of rows or columns in a matrix.

% rowList is a list of pointers to elements in the rows of the matrix.
% colList is a list of pointers to elements in the rows of the matrix.

var rowList : array 1..MaxRowsCols of ^MatrixElement
var colList : array 1..MaxRowsCols of ^MatrixElement

var count   : int  % Number of elements in the matrix
var maxRow  : int  % Maximum row number in which an element is stored.
var maxCol  : int  % Maximum column number in which an element is stored.

% The following are variables used to iterate through the matrix elements.

var nextElem : ^MatrixElement
var currRow   : int

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Accessor operations
%    Return descriptor data for a sparse matrix with no side effects.
%
% Precondition all: The matrix has been initialized.
% Postcondition all: Trivially implied by routine name.

function GetCount()       : int  result count        end GetCount
function GetMaxRow()      : int  result maxRow       end GetMaxRow
function GetMaxCol()      : int  result maxCol       end GetMaxCol
function GetMaxRowsCols() : int  result MaxRowsCols  end GetMaxRowsCols

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set operations
%     There are no set operations for the sparse matrix.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Test operations
%    Used to verify and check the contents of a sparse matrix when new
% operations are added to the class.
%
% PutElemByRow outputs the row, column and data values of the matrix elements
% in row order.  PutElemByCol outputs the row, column and data values of the
% matrix elements in column order.  The procedures can be used to output a
% matrix into a file which can be read in.
%
% Precondition:  Matrix has been initialized.
% Postcondition: All matrix <row,column,data> triplets have been output in
%     row or column order (depending upon which procedure was called).
%     Each item takes up 4 space.  Each triplet has two spaces in between.
%     There are 5 triplets per line.
%     A terminating triplet <0,0,0> its output.
%
% Loop invariant: for i:1..count+1 :: displayedELementTriple(1..i-1)

procedure PutElemByRow()
  var elem : ^MatrixElement   % Element to output
  var count : int := 0       % To put EOL after every 5 elements

  for row : 1 .. maxRow
      elem := rowList(row)
      loop exit when elem = nil
          put elem->GetRow():4, elem->GetCol():4, elem->GetData():4 ..
          count := (count+1) mod 5
          if count = 0 then put ""
                       else put "   " ..
          end if
          elem := elem->GetNextInRow()
      end loop
  end for
  put "   0   0   0"
end PutElemByRow

%%%%%%%%%%%%%%%%%%%%%%%%

procedure PutElemByCol()
  var elem : ^MatrixElement   % Element to output
  var count : int := 0       % To put EOL after every 5 elements

  for col : 1 .. maxCol
      elem := colList(col)
      loop exit when elem = nil
          put elem->GetRow():4, elem->GetCol():4, elem->GetData():4 ..
          count := (count+1) mod 5
          if count = 0 then put ""
                       else put "   " ..
          end if
          elem := elem->GetNextInCol()
      end loop
  end for
  put "   0   0   0"
end PutElemByCol

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GetCountByRow and GetCountByCol count matrix elements by rows and columns
% respectively.  For a correct matrix GetCountByRow = GetCountByCol = GetCount.
%
% Precondition:  The matrix must be initialized.
% Postcondition: verCount = SumCount(rowList(1..maxRow) or
%                verCount = SumCount(colList(1..maxCol)
%
```

```
% Outer loop invariant:
%     for row : 1..maxRow+1 :: verCount = SumCount(rowList(1..row-1))
% or  for col : 1..maxCol+1 :: verCount = SumCount(colList(1..col-1))
%
% Inner loop invariant:
%     for n : 1..maxCol
%         :: verCount = SumCount(rowList(1..row-1)) + SumElemInRow(row,1..n-1)
%  or for n : 1..maxRow
%         :: verCount = SumCount(colList(1..col-1)) + SumElemInCol(col,1..n-1)

function GetCountByRow() : int
  var elem : ^MatrixElement   % Element to count
  var verCount : int := 0     % Verification counter

  for row : 1 .. maxRow
      elem := rowList(row)
      loop exit when elem = nil    % Loop over the elements in a row
          verCount += 1
          elem := elem->GetNextInRow()
      end loop
  end for
  result verCount
end GetCountByRow

%%%%%%%%%%%%%%%%%%%%%%%%%

function GetCountByCol() : int
  var elem : ^MatrixElement   % Element to count
  var verCount : int := 0    % Verification counter

  for col : 1 .. maxCol
      elem := colList(col)
      loop exit when elem = nil    % Loop over the elements in a column
          verCount += 1
          elem := elem->GetNextInCol()
      end loop
  end for
  result verCount
end GetCountByCol

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Utility operations
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterate through the the matrix elements in row order.  Iteration requires
% two operations InitIteration, to start the iteration, and NextIterRowElem,
% to step through the matrix elements.  Both operations make use of the
% support operation SearchRowsFrom.
%
% SearchRowsFrom finds the first matrix element, if any, starting its search
% from fromRow.
%
% Precondition:  Matrix has been initialized and
%                1 <= fromRow <= maxRow
% Postcondition: nextElement points to the first matrix element found and
%                currRow = the row that containing nextElem
%            or  nextElement = nil  and  -- there is no next element
%                currRow = maxRow+1
%
% Loop invariant: for row : fromRow..maxRow+1 :: rowList(1..row-1) = nil

function SearchRowsFrom(fromRow : int) : ^MatrixElement

% Find the first nonempty row from fromRow

    currRow := fromRow
    loop exit when currRow > maxRow or rowList(currRow) ~= nil
        currRow += 1
    end loop

% Set the pointer to the first matrix element, if it exists.

    if currRow <= maxRow then result rowList(currRow) % Element exists
                         else result nil              % Element does not exist
    end if

end SearchRowsFrom

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% InitIteration and NextIterRowElem enable a user to sequentially access
% each matrix element in row order.

% Precondition:  The matrix has been Initialized.
% Postcondition: next element points to first matrix element found  and
%                currRow = the row containing nextElem
%             or nextElem = nil  and  -- matrix is empty
%                currRow = 1

procedure InitIterationByRow()
    nextElem := SearchRowsFrom(1)
end InitIterationByRow

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Tutorial aside: the ’ character is used to identify the new value of a
% variable from its old value, so logical relationships between the old and
% new values can be written.  For example, if you want to add 1 to a variable
% count, then the specification can be written as  count’ = count + 1.
% In the following Postcondition both old and new values of newElem are
% referenced and we need to be able to distinguish which one we mean.  The
% postcondition states that the existing value of newElem is returned but
% before returning that value a new value of newElem is computed.
% End of Tutorial aside.

% Precondition:  Inititeration has been called.
% Postcondition: Result = nextElem  and (
%           -- if there is a next element --
%                nextElem’ = the next element in the matrix and
%                curRow = the row containing nextElem’
%
%        or -- if there is no next element --
%                nextElem’ = nil  and
%                currRow = maxRow +1 )

function NextIterRowElem() : ^MatrixElement

% Check for no elements left in the sequence being iterated,  If there are
% no elements left, tell the caller.

   if nextElem = nil then result nil end if

% At least one more element.  tempElem points to it for return to caller.

    var tempElem : ^MatrixElement := nextElem

% Prepare for the next call to the function by setting nextElem to the next
```

```
% element in the sequence.

    nextElem := nextElem->GetNextInRow()       % Advance across columns
    if nextElem = nil then                     % If end of column
        nextElem := SearchRowsFrom(currRow+1)  % go to a new row
    end if

    result tempElem
end NextIterRowElem

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
include "spmat2.tu"   % Part 2, the rest of the implementation

end SparseMatrix
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File:   spmat2.tu
% Author: Gunnar Gotshalks          Date: 1997 January 21
% Modified by: ??????????           Date: ???????????
%
% This file contains Part 2/2 of the sparse matrix class.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Constructor and destructor operations.
%
% Initialize  must be the first procedure called when using a sparse matrix.
% Initial conditions for an empty matrix are established.
%
% Precondition:  True
% Postcondition: count = maxRow = maxCol = 0 and
%     rowList(1..MaxRowsCols) = nil = colList(1..MaxRowsCols)

procedure Initialize()
end Initialize

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Finalize must be the last procedure called when using a sparse matrix.
% Its purpose is to return all matrix elements to the heap.  It uses direct
% knowledge of the implementation.
%
% Precondition:  The matrix has been initialized.
% Postcondition: All matrix elements are finalized and freed.  The matrix is
%     initialized.

procedure Finalize()
end Finalize

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Utility operations
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Depending upon its input parameters, Insert either inserts a new element,
% replaces the data value for an existing element, or deletes an existing
% element.
%
% Precondition:  Matrix has been initialized.
%       and  1 <= row <= MaxRowsCols
%       and  1 <= col <= MaxRowsCols
%
% Postcondition:
%       if data ~= 0 and  existsElement(row,col) then data replaces the value
%       if data ~= 0 and ~existsElement(row,col) then new element is inserted
%       if data  = 0 and  existsElement(row,col) then the element is deleted
%       if data  = 0 and ~existsElement(row,col) then skip

procedure Insert(row, col, data : int)
end Insert

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GetData returns the value of a data item represented in the array.
% If rowCol = 'r', the search is along a row list.  If rowCol = 'c', the
% search is along a column list.
%
% Precondition:  1 <= row <= maxRow  and  1 <= col <= maxCol
%           and  rowCol in {'r','c'}
% Postcondition:  data at element <row,col>

function GetData(row, col : int, rowCol : char) : int
end GetData

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ExistsElement checks if an element is physically present.
% If rowCol = 'r', the search is along a row list.  If rowCol = 'c', the
% search is along a column list.
%
% Precondition:  1 <= row <= maxRow  and  1 <= col <= maxCol
%           and  rowCol in {'r','c'}
% Postcondition: result = true  => element physically exists
%           and  result = false => element does not physically exist

function ExistsElement(row, col : int, rowCol : char) : boolean
end ExistsElement


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Put prints the sparse matrix in a square matrix with right justified "**"
% for the nonexistent elements.
%
% Precondition:  Matrix has been initialized.
% Postcondition: Matrix is displayed in square format, maxRow x maxCol, with
%     colHeadings(1..maxCol) and rowHeadings(1..maxRow)

procedure Put()
end Put

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Sum adds two matrices -- matrix1 and matrix2.  The algorithm uses a row order
% iterator to scan through each matrix to merge and insert the appropriate
% sum elements.  The caller, self, does not have to be empty at the time of the
% call as this operation finalizes self.
%
% Precondition: self, matrix1 and matrix2 have been initialized.
% Postcondition: self = matrix1 + matrix2

procedure Sum(matrix1, matrix2 : ^SparseMatrix)
end Sum
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File:   spmat.dem
% Author: Gunnar Gotshalks
% Data:   1997 January 21
%
% An interactive demonstration program for sparse arrays.  The commands
% are strings which makes the program not convenient too use interactively
% although the possible.  Effective demonstrations require more input than
% can be reasonable manually entered.  The program is meant more to be used
% on preprepared input files.  The commands are longer to make input files
% self descriptive and the commands themselves easier to remember.
%
% NOTE FOR MAC USERS: to use interactively  you need to enter <return><return>
% after every command.  On Ariel a single <return> works.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

import SparseMatrix in "spmat.tu",
       Debug in "debug.tu"

var cmd : string   % A command is a string so longer commands are possible
                   % in self descriptive input files.

const MaxNumberOfMatrices := 10
var matrix : array 1..MaxNumberOfMatrices of ^SparseMatrix

% Initialize the demonstration.

for i : 1..MaxNumberOfMatrices   % Create the matricies
  new matrix(i)
end for

Debug.Initialize()

% Forward declare support procedures so main program is first.

forward procedure PutHelp()
forward procedure InitMatrix()
forward procedure InsertElements()
forward procedure PutMatrix()
forward procedure AddTwoMatrices()
forward procedure SetFlag()
forward procedure ResetFlag()
forward procedure ToggleFlag()
forward procedure FinalizeMatrix()
forward procedure PutMatrixByRow()
forward procedure PutMatrixByCol()
forward procedure GetStatistics()
forward procedure EchoComment()
forward procedure SkipInput()

% For a comment command we must check for %<return> as a special case.
% The return is read - to determine the end of the string.  If we simply
% read to end of line we pick up the line after %<return> as the comment
% potentially skipping over valid commands.  This means we need to read the
% first character of a command, examine it and if it is not % then read the
% rest of the cmd into the command string.  If is % then we need to read the
% next character to determine if it is the end of line character, in which case
% there is no comment to process.  If there is no end of line character , then
% there is a comment to process.

% Loop invariant: for i: 2..N ::
%                      input(1..N) = cmd(1..i-1) + ch + input(i+1..N)

function GetCommand() : string
  var ch : char
  var cmd : string := ""
  put "Command: " ..
  get skip, ch ; put ch.. ; cmd := cmd + ch
  if ch = '%' then EchoComment()
  else
      get ch
      loop exit when ~(   ('a' <= ch and ch <= 'z')
                       or ('A' <= ch and ch <= 'Z'))
          put ch .. ; cmd := cmd + ch ; get ch
      end loop
  end if
  result cmd
end GetCommand

put "Start of demonstration"

% Loop invariant: processedCommand(1..i-1)  and display is at start of a line.

loop
  cmd := GetCommand()
  case cmd of
    label "Quit"    : put "" ; exit
    label "Init"    : InitMatrix()
    label "Final"   : FinalizeMatrix()
    label "Insert"  : InsertElements()
    label "Put"     : PutMatrix()
    label "Sum"     : AddTwoMatrices()
    label "PutRow"  : PutMatrixByRow()
    label "PutCol"  : PutMatrixByCol()
    label "GetStat" : GetStatistics()
    label "Set"     : SetFlag()
    label "Reset"   : ResetFlag()
    label "Toggle"  : ToggleFlag()
    label "Skip"    : SkipInput()
    label "%"       : % Do nothing, already processed in GetCommand
    label "?","h"   : PutHelp()
    label : put "Unknown command '",cmd,"'"
  end case
end loop

put "End of demonstration"

body PutHelp()
  put ""
  put "  Quit       - to quit the program"
  put "  Init m     - to initialize matrix m"
  put "  Final m    - finialize matrix m"
  put "  Insert m {r c d} 0 0 0    - to insert d at (r,c)"
  put "  Put m      - to show matrix m"
  put "  Sum a b c  -  c := a + b"
  put "  PutRow m   - output matrix m in row order"
  put "  PutCol m   - output matrix m in column order"
  put "  Set f      - Set debug flag f"
  put "  Reset f    - Reset debug flag f"
  put "  Toggle f   - toggle debug flag f"
  put "  GetStat    - Get statistics"
  put "  %          - Comment to end of line"
  put "  ?,h        - to get this listing"
end PutHelp

function GetMatrixNumber() : int
```

```
  var matID : int    % The matrix number in a command
  get matID
  assert (1 <= matID and matID <= MaxNumberOfMatrices)
  put " matrix ", matID ..
  result matID
end GetMatrixNumber

function GetFlagNumber() : int
  var flagID : int   % The flag number in a command
  get flagID
  assert(0 <= flagID and flagID <= Debug.NumberOfFlags-1)
  put " flag ", flagID
  result flagID
end GetFlagNumber

body InitMatrix()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  put ""
  matrix(matID)->Initialize()
end InitMatrix

body FinalizeMatrix()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  put ""
  matrix(matID)->Finalize()
end FinalizeMatrix

body InsertElements()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  var row   : int    % A row number for an element
  var col   : int    % A column number for an element
  var data  : int    % A data value for an element

  put ""
  get row, col, data

% Loop invariant:  insertedTriples(1..i-1)

  loop exit when row = 0
    matrix(matID)->Insert(row,col,data)
    get row, col, data
  end loop
end InsertElements

body PutMatrix()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  put "\n"
  matrix(matID)->Put()
  put ""
end PutMatrix

body AddTwoMatrices()
  var number1, number2, number3 : int   % The matrix numbers in the command
  number3 := GetMatrixNumber()
  put " = " ..
  number1 := GetMatrixNumber()
  put " + " ..
  number2 := GetMatrixNumber()
  put ""
  matrix(number3)->Sum(matrix(number1),matrix(number2))
end AddTwoMatrices

body SetFlag()
  var flagID : int := GetFlagNumber()   % The flag number in a command
  Debug.Set(flagID)
end SetFlag

body ResetFlag()
  var flagID : int := GetFlagNumber()   % The flag number in a command
  Debug.Reset(flagID)
end ResetFlag

body ToggleFlag()
  var flagID : int := GetFlagNumber()   % The flag number in a command
  Debug.Toggle(flagID)
end ToggleFlag

body PutMatrixByRow()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  put ""
  matrix(matID)->PutElemByRow()
end PutMatrixByRow

body PutMatrixByCol()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  put ""
  matrix(matID)->PutElemByCol()
end PutMatrixByCol

body GetStatistics()
  var matID : int := GetMatrixNumber()  % The matrix number in a command
  put ""
  put "Count = ", matrix(matID)->GetCount() ,
      "; Max row = ", matrix(matID)->GetMaxRow() ,
      "; Max column = ",  matrix(matID)->GetMaxCol() ,
      "; Count by row = ", matrix(matID)->GetCountByRow() ,
      "; Count by column = ", matrix(matID)->GetCountByCol()
end GetStatistics

% Loop invariant: input(1..N) = output(1..i-1) + ch + input(i+1..N)

body EchoComment()
  var ch : char
  get ch
  loop exit when ch = '\n'
    put ch ..
    get ch
  end loop
  put ""
end EchoComment

% When reading input from a file this command enables you to bracket
% a segment of input with Skip ... EndSkip.  All input between Skip and
% EndSkip is ignored.  Using this you can develop demonstration input
% but do not have to see all of it while testing later additions.

body SkipInput()
   var aString : string
   loop get aString
      exit when aString = "EndSkip"
   end loop
   put "\n         ... \nCommand: EndSkip\n"
end SkipInput
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File:    sample.in
% Author:  Gunnar Gotshalks
% Date:    1997 January 23
%
% An example demonstration file for spmat.dem.  Every command and its
% effect is shown.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The first sequence of demonstration commands show Init, Insert, and
% Put. It also demonstrate the use of the comment command.
%
% Create matrix 1 and show it

Init 1
Insert 1  1 2 3  4 5 7  8 8 9  2 3 4  5 6 8  8 9 -1  3 4 5
  3 5 6  0 0 0
Put 1
Set 2
Final 1
Reset 2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The second sequence of commands show the use of the debug flags
% using the demonstration commands Set, Reset, and Toggle.
%
% Create matrix 2, set debug flag 0 which in the solution turns on
% debugging for the Insert operation so every insert is recorded.

Init 2  Set 0
Insert 2  2 5 0   3 4 -5  5 3 -6  0 0 0

% Finalize matrix 2 and reinsert the same elements with debug flag 1.
% Now the count, maxRow and maxCol are displayed after the Insert as
% well as the previous debug statements.

Final 2  Set 1
Insert 2  2 5 0   3 4 -5  5 3 -6  0 0 0

% Flag 1 is nested within the scope of flag 2, so when we reset flag 0
% no debugging statements are executed (flag 1 is still true)

Reset 0  Final 2
Insert 2  2 5 0   3 4 -5  5 3 -6  0 0 0

% The final debug flag example toggles flag 0 (making it true since it
% is now false) and toggles flag 1( setting it to false since it is now
% true).  This is same as the the first test in this sequence.

Toggle 0 Toggle 1 Final 2
Insert 2  2 5 0   3 4 -5  5 3 -6  0 0 0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The third sequence of tests demonstrates the commands PutRow and
% PutCol, and GetStat to examine the contents of matrix 2
%
% PutRow outputs matrix 2 in row order -- output is row by row
%
PutRow 2
%
% PutCol outputs matrix 2 in column order -- output column by column
%
PutCol 2
%
% GetStat outputs metatdata counts and maxima about matrix 2
%
GetStat 2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Now to compute the Sum of a couple of matrices.  Rest flag 0 and
% flag 1 and finalize matrices 1 and 2 incase of changes made above
% this point in the file.

Reset 0  Reset 1  Final 1  Final 2

% Recreate matrix 1 and matrix 2

Insert 1  1 2 3  4 5 7  8 8 9  2 3 4  5 6 8  8 9 -1  3 4 5   3 5 6  0 0 0
Insert 2  9 8 7  6 5 4  3 2 1  8 7 6  5 4 3  2 1 0   3 4 -5  3 5 -6  0 0 0

% Create the sum in matrix 3, and display all three matrices.

Init 3  Sum 3 1 2   Put 1  Put 2  Put 3

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The final section shows the rest of the commands

% For Skip, examine sample.in to see what was skipped over.

Skip -- anything that is between here and the end of command (see below)
   is ignored (skipped over)
   the first word on the next line is the end of skip command.
EndSkip

% The set of commands can be displayed in one of two ways.
% The first is with "?"

?

% The second is with "h"

h

% Last of all is the quit command to terminate the demonstrations.
%

Quit
```

```
Start of demonstration
Command: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Command: % File:    sample.in
Command: % Author:  Gunnar Gotshalks
Command: % Date:    1997 January 23
Command: %
Command: % An example demonstration file for spmat.dem.  Every command and its
Command: % effect is shown.
Command: %
Command: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Command: % The first sequence of demonstration commands show Init, Insert, and
Command: % Put. It also demonstrate the use of the comment command.
Command: %
Command: % Create matrix 1 and show it
Command: Init matrix 1
Command: Insert matrix 1
Command: Put matrix 1

         1    2    3    4    5    6    7    8    9
       ___  ___  ___  ___  ___  ___  ___  ___  ___
  1|    **    3   **   **   **   **   **   **   **
  2|    **   **    4   **   **   **   **   **   **
  3|    **   **   **    5    6   **   **   **   **
  4|    **   **   **   **    7   **   **   **   **
  5|    **   **   **   **   **    8   **   **   **
  6|    **   **   **   **   **   **   **   **   **
  7|    **   **   **   **   **   **   **   **   **
  8|    **   **   **   **   **   **   **    9   -1

Command: Set flag 2
Command: Final matrix 1
Count before finalize is 8
Count after Finalize is 0
Command: Reset flag 2
Command: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Command: % The second sequence of commands show the use of the debug flags
Command: % using the demonstration commands Set, Reset, and Toggle.
Command: %
Command: % Create matrix 2, set debug flag 0 which in the solution turns on
Command: % debugging for the Insert operation so every insert is recorded.
Command: Init matrix 2
Command: Set flag 0
Command: Insert matrix 2
  Insert with row = 2; column = 5; data = 0
  Exiting Insert
  Insert with row = 3; column = 4; data = -5
  Exiting Insert
  Insert with row = 5; column = 3; data = -6
  Exiting Insert
Command: % Finalize matrix 2 and reinsert the same elements with debug flag 1.
Command: % Now the count, maxRow and maxCol are displayed after the Insert as
Command: % well as the previous debug statements.
Command: Final matrix 2
Command: Set flag 1
Command: Insert matrix 2
  Insert with row = 2; column = 5; data = 0
Count is 1; maxRow is 2; maxCol is 5
  Exiting Insert
  Insert with row = 3; column = 4; data = -5
Count is 2; maxRow is 3; maxCol is 5
  Exiting Insert
  Insert with row = 5; column = 3; data = -6
Count is 3; maxRow is 5; maxCol is 5
  Exiting Insert
```

```
Command: % Flag 1 is nested within the scope of flag 2, so when we reset flag 0
Command: % no debugging statements are executed (flag 1 is still true)
Command: Reset flag 0
Command: Final matrix 2
Command: Insert matrix 2
Command: % The final debug flag example toggles flag 0 (making it true since it
Command: % is now false) and toggles flag 1( setting it to false since it is now
Command: % true).  This is same as the the first test in this sequence.
Command: Toggle flag 0
Command: Toggle flag 1
Command: Final matrix 2
Command: Insert matrix 2
  Insert with row = 2; column = 5; data = 0
  Exiting Insert
  Insert with row = 3; column = 4; data = -5
  Exiting Insert
  Insert with row = 5; column = 3; data = -6
  Exiting Insert
Command: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Command: % The third sequence of tests demonstrates the commands PutRow and
Command: % PutCol, and GetStat to examine the contents of matrix 2
Command: %
Command: % PutRow outputs matrix 2 in row order -- output is row by row
Command: %
Command: PutRow matrix 2
   2   5   0      3   4  -5      5   3  -6      0   0   0
Command: %
Command: % PutCol outputs matrix 2 in column order -- output column by column
Command: %
Command: PutCol matrix 2
   5   3  -6      3   4  -5      2   5   0      0   0   0
Command: %
Command: % GetStat outputs metatdata counts and maxima about matrix 2
Command: %
Command: GetStat matrix 2
Count = 3; Max row = 5; Max column = 5; Count by row = 3; Count by column = 3
Command: %
Command: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Command: % Now to compute the Sum of a couple of matrices.  Rest flag 0 and
Command: % flag 1 and finalize matrices 1 and 2 incase of changes made above
Command: % this point in the file.
Command: Reset flag 0
Command: Reset flag 1
Command: Final matrix 1
Command: Final matrix 2
Command: % Recreate matrix 1 and matrix 2
Command: Insert matrix 1
Command: Insert matrix 2
Command: % Create the sum in matrix 3, and display all three matrices.
Command: Init matrix 3
Command: Sum matrix 3 =  matrix 1 +  matrix 2
Command: Put matrix 1

         1    2    3    4    5    6    7    8    9
       ___  ___  ___  ___  ___  ___  ___  ___  ___
  1|    **    3   **   **   **   **   **   **   **
  2|    **   **    4   **   **   **   **   **   **
  3|    **   **   **    5    6   **   **   **   **
  4|    **   **   **   **    7   **   **   **   **
  5|    **   **   **   **   **    8   **   **   **
  6|    **   **   **   **   **   **   **   **   **
  7|    **   **   **   **   **   **   **   **   **
  8|    **   **   **   **   **   **   **    9   -1
```

```
Command: Put matrix 2

        1    2    3    4    5    6    7    8
       ___  ___  ___  ___  ___  ___  ___  ___
  1|    **   **   **   **   **   **   **   **
  2|     0   **   **   **   **   **   **   **
  3|    **    1   **   -5   -6   **   **   **
  4|    **   **   **   **   **   **   **   **
  5|    **   **   **    3   **   **   **   **
  6|    **   **   **   **    4   **   **   **
  7|    **   **   **   **   **   **   **   **
  8|    **   **   **   **   **   **    6   **
  9|    **   **   **   **   **   **   **    7

Command: Put matrix 3

        1    2    3    4    5    6    7    8    9
       ___  ___  ___  ___  ___  ___  ___  ___  ___
  1|    **    3   **   **   **   **   **   **   **
  2|     0   **    4   **   **   **   **   **   **
  3|    **    1   **   **   **   **   **   **   **
  4|    **   **   **   **    7   **   **   **   **
  5|    **   **   **    3   **    8   **   **   **
  6|    **   **   **   **    4   **   **   **   **
  7|    **   **   **   **   **   **   **   **   **
  8|    **   **   **   **   **   **    6    9   -1
  9|    **   **   **   **   **   **   **    7   **

Command: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Command: % The final section shows the rest of the commands
Command: % For Skip, examine sample.in to see what was skipped over.
Command: Skip
         ...
Command: EndSkip

Command: % The set of commands can be displayed in one of two ways.
Command: % The first is with "?"
Command: ?
  Quit       - to quit the program
  Init m     - to initialize matrix m
  Final m    - finialize matrix m
  Insert m {r c d} 0 0 0    - to insert d at (r,c)
  Put m      - to show matrix m
  Sum a b c  -  c := a + b
  PutRow m   - output matrix m in row order
  PutCol m   - output matrix m in column order
  Set f      - Set debug flag f
  Reset f    - Reset debug flag f
  Toggle f   - toggle debug flag f
  GetStat    - Get statistics
  %          - Comment to end of line
  ?,h        - to get this listing
Command: % The second is with "h"
Command: h
  Quit       - to quit the program
  Init m     - to initialize matrix m
  Final m    - finialize matrix m
  Insert m {r c d} 0 0 0    - to insert d at (r,c)
  Put m      - to show matrix m
  Sum a b c  -  c := a + b
  PutRow m   - output matrix m in row order
  PutCol m   - output matrix m in column order
  Set f      - Set debug flag f
  Reset f    - Reset debug flag f
  Toggle f   - toggle debug flag f
  GetStat    - Get statistics
  %          - Comment to end of line
  ?,h        - to get this listing
Command: % Last of all is the quit command to terminate the demonstrations.
Command: %
Command: Quit
End of demonstration
```