```
-- File: menus.mesa  Edited by: Johnsson, September 27, 1977  11:33 AM

DIRECTORY
  InlineDefs:  FROM "inlinedefs",
  WindowDefs:  FROM "windowdefs",
  StringDefs:  FROM "stringdefs",
  StreamDefs:  FROM "streamdefs",
  SystemDefs: FROM "systemdefs",
  MenuDefs: FROM "menudefs",
  RectangleDefs: FROM "rectangledefs";

DEFINITIONS FROM  WindowDefs, MenuDefs, RectangleDefs,
   StreamDefs;

Menus: PROGRAM
  IMPORTS RectangleDefs, StreamDefs, StringDefs, SystemDefs, WindowDefs
  EXPORTS MenuDefs, RectangleDefs, WindowDefs SHARES MenuDefs, RectangleDefs =
  BEGIN

-- GLOBAL Data

  defaultpfont: FAptr ← NIL;
  defaultlineheight: INTEGER;
  nullindex:  StreamIndex = StreamIndex[0,-1];
  originindex:  StreamIndex = StreamIndex[0,0];
  bbTable:  ARRAY[0..SIZE[BBTable] + 1] OF WORD;
  bbptr:  BBptr ← LOOPHOLE[@bbTable];
  CR: CHARACTER = 15C;

-- Mesa Display Menu Routines

  CreateMenu: PUBLIC PROCEDURE [array: MenuArray]
    RETURNS[MenuHandle] =
    BEGIN
    -- declare locals
      i, j: CARDINAL;
      menu: MenuHandle;
      width, widest: xCoord;
    -- compute width of widest command word
      [defaultpfont, defaultlineheight] ← GetDefaultFont[];
      widest ← 0;
      FOR i IN [0..LENGTH[array]) DO
        width ← 0;
        FOR j IN [0..array[i].keyword.length) DO
          width ← width + ComputeCharWidth[array[i].keyword[j], defaultpfont];
          ENDLOOP;
        IF width > widest THEN widest ← width;
        ENDLOOP;
    -- now create menu object and init it
      widest ← widest + menuleftmargin*2;
      menu ← SystemDefs.AllocateHeapNode[SIZE[MenuObject]];
      menu↑ ← MenuObject[NIL, -1, widest, NIL, NIL, NIL, array];
    RETURN[menu];
    END;

  DestroyMenu: PUBLIC PROCEDURE [menu: MenuHandle] =
    BEGIN
      -- NOTE: need to unlink from menu list later
      SystemDefs.FreeHeapNode[menu];
    END;

  DisplayMenu: PUBLIC PROCEDURE
    [menu: MenuHandle, mapdata: BMHandle, x: xCoord, y: yCoord]=
    BEGIN
    -- declare locals
      i: CARDINAL;
      cx: INTEGER ← MAX[0, x-(menu.width+2)];
      cy: INTEGER;
      length: CARDINAL = LENGTH[menu.array];
      clearwords: GrayArray ← [0, 0, 0, 0];
      clear: GrayPtr = @clearwords;
      [defaultpfont, defaultlineheight] ← GetDefaultFont[];
    -- save data first then clear it
      IF menu.index = -1 THEN
        cy ← y-(defaultlineheight/2)
      ELSE
```

```
        cy ← y-(menu.index*defaultlineheight+(defaultlineheight/2));
      cy ← MAX[0, cy];
      menu.rectangle ← CreateRectangle[mapdata, cx, menu.width, cy, defaultlineheight*length+2];
      menu.rectangle.options.NoteOverflow ← TRUE;
      --test page size of menu before saving
      IF (((menu.rectangle.cw + 15) /16 +1) *
            menu.rectangle.ch ) / 256 <= 4
        THEN menu.dataseg ← LOOPHOLE[SaveRectangle[menu.rectangle]];
      menu.index ← -1;
      ClearBoxInRectangle[menu.rectangle, 0, menu.rectangle.cw, 0, menu.rectangle.ch, clear];
    -- now paint it up on screen
      DrawBoxInRectangle[menu.rectangle, 0, menu.rectangle.cw, 0, menu.rectangle.ch];
      FOR i IN [0..length) DO
        [cx,cy] ← WriteRectangleString[menu.rectangle, menuleftmargin, (i*defaultlineheight + 1), menu.
**array[i].keyword, defaultpfont
          !
          RectangleError =>
            SELECT error FROM
              RightOverflow => CONTINUE;
              NotVisible ,
              BottomOverflow => EXIT;
              ENDCASE
          ];
        ENDLOOP;
    END;

  ClearMenu: PUBLIC PROCEDURE [menu: MenuHandle] =
    BEGIN
    -- define locals
      clearwords: GrayArray ← [0, 0, 0, 0];
      clear: GrayPtr = @clearwords;
    -- clear it now, repaint it later
      IF menu.rectangle # NIL THEN
        BEGIN
        ClearBoxInRectangle[menu.rectangle, 0, menu.rectangle.cw, 0, menu.rectangle.ch, clear];
        IF menu.dataseg # NIL
          THEN BEGIN
            RestoreRectangle[menu.rectangle, LOOPHOLE[menu.dataseg]];
            menu.dataseg ← NIL;
            END;
        DestroyRectangle[menu.rectangle];
        menu.rectangle ← NIL;
        END;
    END;

  MarkMenuItem: PUBLIC PROCEDURE
    [menu: MenuHandle, index: INTEGER] =
    BEGIN
    -- define locals
      i:  INTEGER ← 1;
      r: Rptr = menu.rectangle;
      [defaultpfont, defaultlineheight] ← GetDefaultFont[];
    -- if same as before then ignore
      IF menu.index = index THEN RETURN;
    -- fix for bottom of menu
      IF menu.index = LENGTH[menu.array]-1 THEN i ← 0;
      IF menu.index # -1 THEN -- turn old guy off
        InvertBoxInRectangle[r, 1, r.cw-2, menu.index*defaultlineheight+1, defaultlineheight-i];
      IF index = LENGTH[menu.array]-1 THEN i ← 0
        ELSE i ← 1;
      IF index # -1 THEN -- turn new guy on
        InvertBoxInRectangle[r, 1, r.cw-2, index*defaultlineheight+1, defaultlineheight-i];
      menu.index ← index;
    END;

SaveRectangle:PUBLIC PROCEDURE[rectangle:  Rptr] RETURNS[POINTER]=
  BEGIN
  --declare locals
  SegPtr:  POINTER;
  wordsperline:  CARDINAL = rectangle.bitmap.wordsperline;
  mapaddr:  BMptr = rectangle.bitmap.addr;
  dw:  CARDINAL ← rectangle.cw;
  dh:  CARDINAL ← rectangle.ch;
  dwWords:  CARDINAL ← (dw + 15)/16 + 1;
  totalWords:  CARDINAL ← dwWords * dh;
  SegPtr ← SystemDefs.AllocateSegment[totalWords];
```

```
  bbptr ← EVEN[bbptr];
  bbptr↑ ← BBTable[0,block,replace,0,SegPtr,dwWords,
     0,0,dw,dh,mapaddr,wordsperline,rectangle.x0,
     rectangle.y0,0,0,0,0];
  BitBlt[bbptr];
  RETURN[SegPtr];
  END;


RestoreRectangle:PUBLIC PROCEDURE[rectangle:  Rptr, SegPtr:  POINTER]=
BEGIN
  --declare locals
  wordsperline:  INTEGER = rectangle.bitmap.wordsperline;
  mapaddr:  BMptr = rectangle.bitmap.addr;
  dw:  CARDINAL ← rectangle.cw;
  dh:  CARDINAL ← rectangle.ch;
  dwWords:  CARDINAL ← (dw + 15)/16 + 1;
  bbptr ← EVEN[bbptr];
  bbptr↑ ← BBTable[0,block,replace,0,mapaddr,wordsperline,
     rectangle.x0,rectangle.y0,dw,dh,SegPtr,
     dwWords,0,0,0,0,0,0];
  BitBlt[bbptr];
  SystemDefs.FreeSegment[SegPtr];
  RETURN;
  END;

EVEN:  PROCEDURE[v:  UNSPECIFIED] RETURNS[UNSPECIFIED]=
  BEGIN
  RETURN[v+ InlineDefs.BITAND[v,1]];
  END;

-- Selection routines

ResolveBugToPosition: PUBLIC PROCEDURE [w: WindowHandle, x: xCoord, y: yCoord]
  RETURNS[INTEGER, xCoord, INTEGER, StreamIndex]=
   BEGIN
   -- Declare Locals
   char: CHARACTER;
   xpos: xCoord ← leftmargin;
   width:  CARDINAL ← 0;
   nlines, line: CARDINAL;
   index, savedindex: StreamIndex;
   -- NOTE: following array is ONE origin
   linestarts: DESCRIPTOR FOR ARRAY OF StreamIndex;
   nlines ← (w.rectangle.ch/w.ds.lineheight)-1;
   linestarts ← DESCRIPTOR[GetLineTable[], nlines +1];
   IF EqualIndex[linestarts[0] ,nullindex] THEN -- empty window
     RETURN[1, leftmargin, 0, originindex];
   savedindex ← GetIndex[w.file];
   [x, y] ← CursorToRecCoords[w.rectangle, x, y];
   -- NOTE:  real line number = line + 1
   line ← MIN[MAX[1, y/w.ds.lineheight], nlines] - 1;
   -- back up until real text in window
   UNTIL NOT EqualIndex[linestarts[line],nullindex] DO
     line ← line-1;
     ENDLOOP;
   index ← linestarts[line];
   SetIndex[w.file, index];
   WHILE x > xpos DO
     index ← GetIndex[w.file];
     char ← w.file.get[w.file
       ! StreamError => EXIT];
     width ← IF char = 11C THEN ComputeTabWidth[w.ds.pfont,xpos]
       ELSE ComputeCharWidth[char, w.ds.pfont];
     xpos ← xpos + width;
     IF EqualIndex[index, w.eofindex] OR char = CR OR
        EqualIndex[GetIndex[w.file],linestarts[line +1]]
       THEN EXIT;
     ENDLOOP;
   SetIndex[w.file, savedindex];
   RETURN[line+1, xpos-width, width, index];
   END;

  ComputeTabWidth: PROCEDURE [font: FAptr, x: xCoord]
    RETURNS [CARDINAL] =
    BEGIN
```

```
    tw: CARDINAL = ComputeCharWidth[' ,font] * 8;
    RETURN[tw - x MOD tw]
    END;

  GetSelection: PUBLIC PROCEDURE [w: WindowHandle]
    RETURNS[STRING]=
    BEGIN
    -- Declare Locals
      count: CARDINAL;
      savedindex: StreamIndex;
      str: STRING;
    -- put the selection into a string
      IF EqualIndex[w.selection.rightindex, nullindex]
        THEN RETURN[str ← NIL];
      savedindex ← GetIndex[w.file];
      count ← (w.selection.rightindex.page-w.selection.leftindex.page)*512 +
      (w.selection.rightindex.byte+1)-w.selection.leftindex.byte;
      str ← SystemDefs.AllocateHeapString[count];
      SetIndex[w.file, w.selection.leftindex];
      THROUGH [0..count) DO
        StringDefs.AppendChar[str, w.file.get[w.file]];
        ENDLOOP;
      SetIndex[w.file, savedindex];
      RETURN[str];
    END;


MakeSelection: PUBLIC PROCEDURE
    [w: WindowHandle, sel: POINTER TO Selection] =
    BEGIN
    -- unmark the old one if one exists and is visible
      IF  NOT (EqualIndex[w.selection.leftindex,nullindex]
        OR w.selection.leftline = 0) THEN MarkSelection[w];
    -- mark the new one
      w.selection ← sel↑;
      MarkSelection[w];
    END;

MarkSelection: PUBLIC PROCEDURE
    [w: WindowHandle] =
    BEGIN
    -- Declare Locals
      i: CARDINAL;
    -- check for visibility
        IF EqualIndex[w.selection.leftindex,nullindex]
              OR w.selection.leftline = 0 THEN RETURN;
    -- mark the new one
        IF w.selection.leftline = w.selection.rightline THEN
          InvertBoxInRectangle[w.rectangle, w.selection.leftx, w.selection.rightx-w.selection.leftx,
          w.selection.leftline*w.ds.lineheight,
          w.ds.lineheight]
        ELSE
          BEGIN
          InvertBoxInRectangle[w.rectangle, w.selection.leftx, (w.rectangle.cw-1)-w.selection.leftx,
          w.selection.leftline*w.ds.lineheight,
          w.ds.lineheight];
          i ← w.selection.rightline-w.selection.leftline;
          IF i > 1 THEN
            InvertBoxInRectangle[w.rectangle, leftmargin, w.rectangle.cw-(leftmargin+1), (w.selection.l
**eftline+1)*w.ds.lineheight,
            (i-1)*w.ds.lineheight];
          InvertBoxInRectangle[w.rectangle, leftmargin, w.selection.rightx-leftmargin, w.selection.righ
**tline*w.ds.lineheight,
          w.ds.lineheight];
          END
    END;

UpdateSelection:  PUBLIC PROCEDURE[w:  WindowHandle] =
  BEGIN
  --local data
  which:  BOOLEAN ← TRUE;
  lastindex:  StreamIndex ← nullindex;
  savedindex:  StreamIndex;
  line:  CARDINAL;
  nlines:  CARDINAL ← (w.rectangle.ch/w.ds.lineheight)  - 1;
  i:  CARDINAL ← 0;
```

```
    linestarts : DESCRIPTOR FOR ARRAY OF StreamIndex;
    linestarts ← DESCRIPTOR[GetLineTable[], nlines +1];
    --figure out the real last line
    FOR line IN [1..nlines) DO
          IF EqualIndex[nullindex,linestarts[line]] THEN
            BEGIN
            lastindex ← w.eofindex;
            nlines ← line-1;
            EXIT;
            END;
          ENDLOOP;
    IF EqualIndex[lastindex, nullindex] THEN
      lastindex ← linestarts[nlines];
    -- for no selection or out of bounds
    IF EqualIndex[w.selection.leftindex, nullindex] OR
     GrEqualIndex[w.selection.leftindex, lastindex] OR
     GrIndex[linestarts[0], w.selection.rightindex] THEN
      BEGIN
      w.selection.leftline ← 0;
      RETURN;
      END;
    savedindex ← GetIndex[w.file];
    -- find line numbers
    IF GrEqualIndex[w.selection.leftindex, linestarts[0]] THEN
      FOR i ← 0, i+1 UNTIL i = nlines DO
        IF (GrEqualIndex[w.selection.leftindex, linestarts[i]]
        AND GrIndex[linestarts[i+1], w.selection.leftindex])
        THEN EXIT; ENDLOOP;
    w.selection.leftline ← MAX[1, i+1];
    IF GrIndex[lastindex, w.selection.rightindex] THEN
      FOR i ← i , i+1 UNTIL i = nlines DO
        IF (GrEqualIndex[w.selection.rightindex, linestarts[i]]
        AND GrIndex[linestarts[i+1], w.selection.rightindex])
        THEN EXIT; ENDLOOP;
    w.selection.rightline ← MIN[nlines+1, i+1];
    --find xcoords
    IF GrEqualIndex[w.selection.leftindex, linestarts[0]] THEN
      w.selection.leftx ←
        GetPos[w, linestarts[w.selection.leftline-1],
                w.selection.leftindex, which]
      ELSE w.selection.leftx ← leftmargin;
    which ← FALSE;
    IF GrIndex[lastindex, w.selection.rightindex] THEN
      w.selection.rightx ←
        GetPos[w, linestarts[w.selection.rightline-1],
                w.selection.rightindex, which]
      ELSE w.selection.rightx ← w.rectangle.cw;
    SetIndex[w.file, savedindex];
    MarkSelection[w];
    END;

  GetPos:  PROCEDURE[w:  WindowHandle, index1:  StreamIndex,
    index2:  StreamIndex, left:  BOOLEAN] RETURNS[xCoord]=
    BEGIN
    char:  CHARACTER;
    xpos:  xCoord ← leftmargin;
    width: CARDINAL;
    IF EqualIndex[index1, index2] AND left THEN RETURN[xpos];
    SetIndex[w.file,index1];
    DO index1 ← GetIndex[w.file];
      char ← w.file.get[w.file];
      width ← IF char = 11C THEN ComputeTabWidth[w.ds.pfont,xpos]
        ELSE ComputeCharWidth[char, w.ds.pfont];
      xpos ← xpos + width;
      IF EqualIndex[index1,index2] THEN EXIT;
      ENDLOOP;
    IF left THEN xpos ← xpos - width;
    RETURN[xpos];
    END;

  END. of Menus
```