```
;   EXAM4 2019 - MOUSE DROPPINGS
;   Myles Cruz

Comment!
     This program uses int33h to attach to the system mouse
     driver to control mouse movements.

     Actions for mouse events are supplied by the student as
     per EXAM4.DOC requirements. Refer to the external
     documentation in EXAM4DOC.DOCX for exceptions and
     enhancements. Non implemented components still under
     construction are identified in the external documentation
     EXAM4DOC.DOC and marked with comments in UPPERCASE within
     the modules below.
     A separate file, EXAM4A.asm may contain an alternate
     version of your program with code that "almost" works.

     Assistance received:  (be very specific)

     I am signing this document to verify that I
     have followed the Academic Honor Code without exception.


         HONOR CODE SIGNATURE: Myles Cruz DATE:12/2/2019

       !

; Set up the stack
SSEG    SEGMENT   PARA  STACK  'STACK'
    DB  551  DUP ('MY_STACK')   ; can find the stack under debug
SSEG    ENDS


CSEG    SEGMENT  PARA  PUBLIC
    ; Tell TASM about our segment definitions.
    ASSUME  CS:CSEG, DS:CSEG, SS:SSEG

    ; We are linking BIOS + INOUT10
    ;place the EXTRN statements here
    EXTRN clrscr:near
    EXTRN getch:near, kbhit:near
    EXTRN putCstring10:near
    EXTRN putDec10:near
    EXTRN writeattr:near
    EXTRN READATTR:near
    EXTRN SETCURSOR:near
    EXTRN GETCURSOR:near

; Data placed with the code so that the mouse driver can easily find it.

;Characters and Strings
error               db 'Cannot initialize mouse -- shutting down'
                    db 0
hi                  db 'hi',0           ; used by dcp for debugging

CrapChar            db ' '              ; character used for droppings
                                        ;changed in the main
CrapColor           db 0F0h             ; color used for droppings
                                        ; you will change this
colorBox            db '     ',0        ; string to print out ColorBar
cover               db '  ',0           ; used to cover previous mouseX/mouseY and
previous CrapChar
backgroundBox       db ' ',0            ; used to print out
CrapLabel           db 'CrapChar: ', 0  ; label for CrapChar
xyLabel             db 'mouseX:   mouseY:  ',0  ; print out labels for x and y
coordinates
MenuLabel           db 'Exit( ) Erase( ) Clear( ) Foreground( ) Background( )',0 ;
print out menu line
;Variables
```

```
67  mouseX              db  0   ; horizontal value for mouse pointer
68  mouseY              db  0   ; vertical value for mouse pointer
69  Xcoordinate         db  7   ; location of mouseX label
70  Ycoordinate         db  17  ; location of mouseY label
71  XYRow               db  24  ; location of xyLabel
72  exitButton          db  5   ; location of exit button
73  eraseButton         db  14  ; location of erase button
74  clearButton         db  23  ; location of clear button
75  foregroundButton    db  37  ; location of foreground button
76  backgroundButton    db  51  ; location of background button
77  toggleButton        db  254 ; value of filled in square to identify a highlighted button
78  EraseColor          db  00h ; color black for erasing screen
79  BackgroundColor     db  4Fh ; background color (red background with white foreground)
80  charFlag            db  0   ; used to check if a character was entered
81  moveFlag            db  0   ; used to check if mouse moved
82  leftClickFlag       db  0   ; used to check if left button was pressed
83  rightClickFlag      db  0   ; used to check if right button was pressed
84  eraserFlag          db  0   ; used to check if "Eraser" button was pressed
85  foregroundFlag      db  0   ; used to check if "Foreground" button was pressed
86  backgroundFlag      db  1   ; used to check if "Background" button was pressed
87                              ; set to 1 because, background is originally highlighted
88  exitFlag            dw  0       ; bool used to exit program - set by getch
89                                  ; or mouse
90  ; DATA ENDS HERE
91
92  MAIN: ; Life starts here.
93      MOV   AX,CSEG   ; Make DS point to OUR code segment.
94      MOV   DS,AX     ; Code and data live in one segment.
95
96      call clrscr
97      mov  ax, 00h    ; initialize mouse
98      int  33h        ; call mouse driver
99      cmp  ax, 00h    ; mouse avail?
100     jne   MAINinstalled
101
102     mov  SI, offset error  ; problems capturing our rodent
103     call putCstring10
104     jmp  MAINquit
105
106 MAINinstalled:      ; OK the mouse is allocated to our window.
107     mov  ax, 01h    ; Let's make the little rat visible.
108     int  33h
109
110     ; Good place to get your menu and status bars on the screen
111     CALL DisplayMenu
112     CALL DisplayColors
113     CALL DisplayCrapCharacter
114
115     ; Now we install our mouse event handler.
116     ; When the mouse does anything as described in CX below,
117     ; the driver will automatically call our MouseEvent callback function
118
119         ;YOU NEED TO CHANGE CX IN ORDER TO HANDLE CLICKS
120         mov  CX, 000Fh  ; call mouse event if moved
121             ; Look at the mask in the documentation
122             ; for AX in MouseEvent.
123         push CS
124         pop  ES                     ; ES must point to our CSEG
125         mov  DX, offset MouseEvent  ; DX points to MouseEvent
126         mov  AX, 0Ch                ; Install our interrupt handler
127         int  33h                    ; for mouse events.
128         ; From this point on, the function MouseEvent will be called
129         ; based on the CX mask.
130
131       ; We change the CrapChar through a busy-wait loop.
132       ; This is called polling. We keep asking about a key pressed.
133       ; Compare this to the fcn MouseEvent which is called by the mouse interrupt.
134
135 MAINagain:
```

```
    cmp [exitFlag], 0
    jne  MAINexit    ; exit program
    CMP [charFlag], 0
    JG CrapCharacter    ; if Flag is 1, display new character
    CMP [moveFlag], 0
    JG  printCoordinate ;if Flag is 1, display new coordinate
    CMP [leftClickFlag],0
    JG  leftButton ;if Flag is 1, handle click within function
    CMP [rightClickFlag],0
    JG  rightButton ; if Flag is 1, switch the color attributes
    JMP MAINcheck
printCoordinate:
    CALL DisplayXYCoordinates
    JMP MAINagain
leftButton:
    CALL LeftClick
    JMP MAINagain
rightButton:
    CALL RightClick
    JMP MAINagain
CrapCharacter:
    MOV [CrapChar], AL
    CALL DisplayCrapCharacter   ; display the updated CrapChar
    MOV [charFlag], 0

MAINcheck:
    call kbhit       ; check to see if we have a key
    jz  MAINagain
    MOV [charFlag], 1 ; mark flag if character was entered
    call getch       ; if so, remove the char from the buffer
    cmp  al, 27      ; if ESC, we want to exit as well
    jne   CrapCharacter ; if not ESC, display character
    mov   [exitFlag], 1  ; if ESC set the flag to leave
    jmp MAINagain

MAINexit:           ; shutting down
    MOV DH, 0       ; put toggle on exit button
    MOV DL, 5
    CALL SETCURSOR
    MOV AL, toggleButton
    MOV CX, 1
    MOV BL, BackgroundColor
    CALL writeattr
    mov  ax, 02h    ; hide mouse pointer
    int  33h
    mov  ax, 00h    ; disconnect our mouse handler
    int  33h

MAINquit:
    MOV   AX,4C00h              ; Return control to DOS.
    INT   21h                   ; End of MAIN program.
; ************************************************************
;************************************************************
MouseEvent  Proc  FAR
Comment  !
This function is called by the mouse Interrupt Service Routine (driver).
Make sure that you don't take up too much CPU time in here.

Input parameters:

Note that the actual mouse driver preserves the following registers for us,
so they will not be changed back in the main program.

    AX = events that occurred, depending on mask:
       bit 0 = mouse pointer moved
       bit 1 = left button pressed
       bit 2 = left button released
       bit 3 = right button pressed
       bit 4 = right button released
```

```
205         bit 5 = center button pressed
206         bit 6 = center button released
207
208
209      BX = Current button state:
210         bit 0 = Left button (0 = up, 1 = pressed down)
211         bit 1 = Right button (0 = up, 1 = pressed down)
212         bit 2 = Center button (0 = up, 1 = pressed down)
213
214      CX = Horizontal  coordinate of mouse
215      DX = Vertical  coordinate
216
217          These are used to check how far mouse moved since we were last
218          in MouseEvent.
219      SI = Last vertical mickey count
220      DI = Last horizontal mickey count
221
222      DS = Data seg of the mouse driver. I will reset it to your data seg.
223
224      USE only BIOS interrupts.  NO NOT use any DOS interrupts like getChar and putChar.
225    !
226
227      push ds
228      push ax
229      push dx
230      push cx
231      push bx
232
233         ; data for this driver will be in our code segment.
234      push cs
235      pop  ds   ; ds now points to our code segment
236
237         ; test for events in the order you want priority
238         ; this is basically like a switch-case statement
239
240         ; I'll test for a mouse move for you
241   MEtestmove:
242      cmp  ax, 1        ; the mouse moved
243      JL  MEret
244      shr  cx, 1       ; 8 pixels per char position
245      shr  cx, 1
246      shr  cx, 1
247      shr  dx, 1
248      shr  dx, 1
249      shr  dx, 1
250      mov  [mouseX], cl ; save the new position
251      mov  [mouseY], dl
252      MOV [moveFlag], 1 ; indicates the mouse moved
253
254   MEtestright:
255      CMP AX, 0002h   ; compare bit 1 to see if left button was pressed
256      JNE MErightClick
257      CMP BX, 0001h   ; compare bit 0 to see if left button is pressed or released
258      JNE MErightClick
259      MOV [leftClickFlag], 1
260
261   MErightClick:
262      CMP AX, 0008h   ; compare bit 3 to see if right button was pressed
263      JNE MEret
264      CMP BX, 0002h   ; compare bit 1 to see if right button is pressed or released
265      JNE MEret
266      MOV [rightClickFlag], 1
267
268   MEret:
269      pop bx
270      pop cx
271      pop dx
272      pop ax
273      pop ds
```

```
274         ret             ; back to the mouse driver (ISR)
275     MouseEvent endP
276     ;*************************************************************
277     ;*************************************************************
278     ; My Functions
279     ; Display Functions
280     DisplayMenu proc
281         PUSH SI
282         PUSH AX
283         PUSH BX
284         PUSH CX
285         PUSH DX
286         ;displayed the menu/status lines and labels on both
287         MOV DH, 0
288         MOV DL, 0
289         MOV BL, BackgroundColor
290     DMmenuLines:    ; loop to display menu lines
291         CALL SETCURSOR
292         MOV AL, backgroundBox
293         MOV CX, 80
294         CALL writeattr
295         INC DH
296         CMP DH, 5
297         JL DMmenuLines
298
299         CALL SETCURSOR
300         MOV AL, 196 ; character is a long dash to show end of menu lines
301         MOV CX, 80
302         CALL writeattr
303
304         MOV DH, 24
305     DMstatusLines: ; loop to display status lines
306         CALL SETCURSOR
307         MOV AL, backgroundBox
308         MOV CX, 80
309         CALL writeattr
310         DEC DH
311         CMP DH,22
312         JG DMstatusLines
313
314         CALL SETCURSOR
315         MOV AL, 196 ; character is a long dash to show end of status lines
316         MOV CX, 80
317         CALL writeattr
318
319     DMmenuLabels:
320         MOV DH, 24
321         MOV DL, 0
322         CALL SETCURSOR
323         MOV SI, offset xyLabel ; prints out mouseX and mouseY labels
324         CALL putCstring10
325
326         MOV DH, 0
327         CALL SETCURSOR
328         MOV SI, offset MenuLabel ; prints out menu labels
329         CALL putCstring10
330         MOV DL, backgroundButton
331         CALL SETCURSOR
332         MOV AL, toggleButton
333         MOV CX, 1
334         CALL writeattr
335
336         MOV DH, 23
337         MOV DL, 0
338         CALL SETCURSOR
339         MOV SI, offset CrapLabel ; prints out label for current CrapChar
340         CALL putCstring10
341
342     DisplayMenuReturn:
```

```
343         POP DX
344         POP CX
345         POP BX
346         POP AX
347         POP SI
348         RET
349     DisplayMenu endp
350
351     DisplayColors proc
352         PUSH SI
353         PUSH BX
354         PUSH CX
355         PUSH DX
356         ;displayed the ColorBar
357         MOV BL, 00h ; start with black
358         MOV CX, 1
359         MOV DH, 2
360         MOV DL, 1
361     DCnextColor:
362         CALL SETCURSOR
363         MOV SI, offset colorBox
364         CALL putCstring10
365         ; explained how this works in EXAM4DOC
366         ADD BL, 1
367         PUSH CX
368         MOV CX, 4
369     DCshiftLeft:
370         SHL BL, 1
371         LOOP DCshiftLeft
372         POP CX
373         MOV BH, BL
374         PUSH CX
375         MOV CX, 4
376     DCshiftRight:
377         SHR BH, 1
378         LOOP DCshiftRight
379         POP CX
380         OR BL, BH
381         INC CX
382         ADD DL, 5
383         CMP CX, 8
384         JLE DCcontinueRow
385         CMP CX, 9
386         JG DCcontinueRow
387         INC DH      ; move the different colors' intensities into second row
388         MOV DL, 1
389     DCcontinueRow:
390         CMP CX, 16 ; check if loop ran for all 16 colors
391         JLE DCnextColor
392
393     DisplayColorsReturn:
394         POP DX
395         POP CX
396         POP BX
397         POP SI
398         RET
399     DisplayColors endp
400
401     DisplayXYCoordinates proc
402         PUSH SI
403         PUSH BX
404         PUSH DX
405         ;use a block with background color to cover up previous coordinates
406         MOV DH, XYRow
407         MOV DL, Xcoordinate
408         CALL SETCURSOR
409         MOV BL, BackgroundColor
410         MOV SI, offset cover ;cover up previous X coordinate
411         CALL putCstring10
```

```
412         MOV DH, XYRow
413         MOV DL, Ycoordinate
414         CALL SETCURSOR
415         MOV BL, BackgroundColor
416         MOV SI, offset cover ;cover up previous Y coordinate
417         CALL putCstring10
418         ;print new coordinate
419         MOV DH, XYRow
420         MOV DL, Xcoordinate
421         CALL SETCURSOR
422         MOV DH, 0
423         MOV DL, [mouseX]
424         CALL putDec10
425         MOV DH, XYRow
426         MOV DL, Ycoordinate
427         CALL SETCURSOR
428         MOV DH, 0
429         MOV DL, [mouseY]
430         CALL putDec10
431         MOV [moveFlag],0    ;set back to 0 which means mouse is no longer moving
432
433     DisplayXYCoordinatesReturn:
434         POP DX
435         POP BX
436         POP SI
437         RET
438     DisplayXYCoordinates endp
439
440     DisplayCrapCharacter proc
441         PUSH AX
442         PUSH BX
443         PUSH CX
444         PUSH DX
445         ; display current CrapChar value
446         MOV DH, 23
447         MOV DL, 9
448         CALL SETCURSOR
449         MOV AL, CrapChar
450         MOV CX, 1
451         MOV BL, CrapColor
452         CALL writeattr
453     DisplayCrapCharacterReturn:
454         POP DX
455         POP CX
456         POP BX
457         POP AX
458         RET
459     DisplayCrapCharacter endp
460
461     ; Interrupt Functions
462     LeftClick proc
463         PUSH AX
464         PUSH BX
465         PUSH CX
466         PUSH DX
467         ; checks what was clicked on and sends it to that function
468     LCexit:
469         ; checking if exit button was pressed
470         CALL Exit
471         CMP [exitFlag], 1
472         JNE LCclearScreen
473         JMP LeftClickReturn
474     LCclearScreen:
475         ; check if clear button was pressed
476         CALL ClearScreen
477     LCeraser:
478         ; checking if erase button was pressed
479         CALL Eraser
480         CMP [eraserFlag], 1
```

```
481         JNE LCforegroundBackground
482         JMP LCcrap
483     LCforegroundBackground:
484         ; used to toggle between foreground and background buttons
485         CALL ForegroundBackground
486     LCcolorBar:
487         ; checking to see if color was chosen
488         CALL ColorBar
489     LCcrap:
490         ; clicking on screen to print out CrapChar
491         CALL PrintCrap
492     LeftClickReturn:
493         MOV [leftClickFlag], 0
494         ; set flag back to 0 to indicate left button is no longer clicked
495         POP DX
496         POP CX
497         POP BX
498         POP AX
499         RET
500     LeftClick endp
501
502     RightClick proc
503         PUSH AX
504         PUSH BX
505         PUSH CX
506         ; switches the foreground and background
507         MOV BL, CrapColor
508         MOV BH, BL
509
510         MOV CX, 4
511     RCshiftLeft:
512         SHL BL, 1
513         LOOP RCshiftLeft
514
515         MOV CX, 4
516     RCshiftRight:
517         SHR BH, 1
518         LOOP RCshiftRight
519
520         OR BL, BH
521         MOV CrapColor, BL
522         MOV AL, CrapChar
523         CALL DisplayCrapCharacter
524         MOV [rightClickFlag], 0
525
526     RightClickReturn:
527         POP CX
528         POP BX
529         POP AX
530         RET
531     RightClick endp
532
533     ; Functions
534     PrintCrap proc
535         PUSH AX
536         PUSH BX
537         PUSH CX
538         PUSH DX
539         ;prints out mouse droppings
540         CMP [mouseY], 5 ; checking if clicked outside menu lines
541         JLE PrintCrapReturn
542         CMP [mouseY], 22 ; checking if clicked outside status lines
543         JGE PrintCrapReturn
544         MOV DH, [mouseY]
545         MOV DL, [mouseX]
546         CALL SETCURSOR
547
548         MOV AL, CrapChar
549         MOV CX, 1
```

```
    CMP [eraserFlag], 1 ; if eraser button was pressed
    JE  PCprintErase
    MOV BL, CrapColor   ; makes mouse droppings black, to "erase" previous droppings
    JMP PCprint
PCprintErase:
    MOV BL, EraseColor
PCprint:
    CALL writeattr

PrintCrapReturn:
    POP DX
    POP CX
    POP BX
    POP AX
    RET
PrintCrap endp

Exit proc
    ; checks if "Exit" button was pressed
    CMP [mouseY], 0
    JNE ExitReturn
    CMP [mouseX], 5
    JNE ExitReturn
    MOV [exitFlag], 1
ExitReturn:
    RET
Exit endp

Eraser proc
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    ; checks if "Erase" button was pressed
EcheckEraserCoordinate:
    CMP [mouseY], 0
    JE  EcheckX
    JMP EraserReturn
EcheckX:
    CMP [mouseX], 14
    JE  EcheckEraserFlag
    JMP EraserReturn
EcheckEraserFlag:
    CMP [eraserFlag], 1
    JNE EsetEraserFlag
    MOV [eraserFlag], 0     ; turn off eraser button
    MOV DH, 0
    MOV DL, 14
    CALL SETCURSOR
    MOV AL, cover
    MOV CX, 1
    MOV BL, BackgroundColor
    CALL writeattr          ; covers toggle button if unclicked
    JMP EraserReturn
EsetEraserFlag:
    MOV [eraserFlag], 1     ; turn on eraser button
    MOV DH, 0
    MOV DL, 14
    CALL SETCURSOR
    MOV AL, toggleButton
    MOV CX, 1
    MOV BL, BackgroundColor ; places toggle button if clicked
    CALL writeattr

EraserReturn:
    POP DX
    POP CX
    POP BX
    POP AX
```

```
619         RET
620     Eraser endp
621
622     ClearScreen proc
623         ; clears the "screen" in between menu and status
624         PUSH AX
625         PUSH BX
626         PUSH CX
627         PUSH DX
628         ; checks if "Clear" button was pressed
629         CMP [mouseY], 0
630         JE  CScheckX
631         JMP ClearScreenReturn
632     CScheckX:
633         CMP [mouseX], 23
634         JE CSclearScreen
635         JMP ClearScreenReturn
636         ; places a black dropping over each location
637     CSclearScreen:
638         MOV DH, 6
639         MOV DL, 0
640     CSclearing:
641         CALL SETCURSOR
642         MOV AL, ' '
643         MOV CX, 80
644         MOV BL, EraseColor
645         CALL writeattr
646         INC DH
647         CMP DH, 21
648         JL  CSclearing
649
650     ClearScreenReturn:
651         POP DX
652         POP CX
653         POP BX
654         POP AX
655         RET
656     ClearScreen endp
657
658     ForegroundBackground proc
659         ; checks the toggling between foreground and background buttons
660         PUSH AX
661         PUSH BX
662         PUSH CX
663         PUSH DX
664         ; check if foreground button is clicked and on/off
665         CMP [mouseY], 0
666         JNE FBbackground
667         CMP [mouseX], 37
668         JNE FBbackground
669         CMP [foregroundFlag], 1
670         JNE FBsetForegroundFlag
671         JMP ForegroundBackgroundReturn
672     FBsetForegroundFlag:
673         ; turn background button off and foreground button on
674         MOV [foregroundFlag], 1
675         MOV [backgroundFlag], 0
676         MOV DH, [mouseY]
677         MOV DL, [mouseX]
678         CALL SETCURSOR
679         MOV AL, toggleButton
680         MOV BL, BackgroundColor
681         MOV CX, 1
682         CALL writeattr  ; highlight foreground button
683         MOV DH, 0
684         MOV DL, backgroundButton
685         CALL SETCURSOR
686         MOV AL, cover
687         CALL writeattr  ; cover background button
```

```
688         JMP ForegroundBackgroundReturn
689
690     FBbackground:
691         ;check if background button is clicked and on/off
692         CMP [mouseY],0
693         JNE ForegroundBackgroundReturn
694         CMP [mouseX], 51
695         JNE ForegroundBackgroundReturn
696         CMP [backgroundFlag], 1
697         JNE FBsetBackgroundFlag
698         JMP ForegroundBackgroundReturn
699     FBsetBackgroundFlag:
700         ; turn background button on and foreground button off
701         MOV [backgroundFlag], 1
702         MOV [foregroundFlag], 0
703         MOV DH, [mouseY]
704         MOV DL, [mouseX]
705         CALL SETCURSOR
706         MOV AL, toggleButton
707         MOV BL, BackgroundColor
708         MOV CX, 1
709         CALL writeattr  ; highlight background button
710         MOV DH, 0
711         MOV DL, foregroundButton
712         CALL SETCURSOR
713         MOV AL, cover
714         CALL writeattr  ; cover foreground button
715
716     ForegroundBackgroundReturn:
717         POP DX
718         POP CX
719         POP BX
720         POP AX
721         RET
722     ForegroundBackground endp
723
724     ColorBar proc
725         ; check which color was clicked
726         PUSH AX
727         PUSH BX
728         PUSH DX
729         ; check if clicked within the Color Bar range
730     CBcheckYlow:
731         CMP [mouseY],2
732         JGE CBcheckYhigh
733         JMP ColorBarReturn
734     CBcheckYhigh:
735         CMP [mouseY],3
736         JLE CBcheckXlow
737         JMP ColorBarReturn
738     CBcheckXlow:
739         CMP [mouseX], 1
740         JGE CBcheckXhigh
741         JMP ColorBarReturn
742     CBcheckXhigh:
743         CMP [mouseX], 41
744         JL CBchooseColor
745         JMP ColorBarReturn
746     CBchooseColor:
747         MOV DH, [mouseY]
748         MOV DL, [mouseX]
749         CALL SETCURSOR
750         CALL READATTR   ; stores color in the AH
751         ; checks which button is highlighted
752         CMP [backgroundFlag], 1
753         JE CBbackgroundChange
754         CMP [foregroundFlag], 1
755         JE CBforegroundChange
756     CBbackgroundChange:
```

```
757         MOV BH, CrapColor
758         AND AH, 0F0h
759         AND BH, 0Fh
760         OR  AH, BH
761         JMP CBsetChange
762     CBforegroundChange:
763         MOV BH, CrapColor
764         AND AH, 0Fh
765         AND BH, 0F0h
766         OR  AH, BH
767     CBsetChange:
768         MOV CrapColor, AH   ; move updated color back into CrapColor
769         MOV BL, CrapColor
770         MOV AL, CrapChar
771         CALL DisplayCrapCharacter
772
773     ColorBarReturn:
774         POP DX
775         POP BX
776         POP AX
777         RET
778     ColorBar endp
779
780     CSEG    ENDS            ; End of code segment.
781
782     END     MAIN            ; End of program. Start execution at MAIN
```