Bouncy Ball Simulator

Written by Zachary DeStefano
Email: zackattacknyu@gmail.com

Report submitted in partial fulfillment of the requirements
of the Game Program Final Semester Project
Computer Systems Organization (Honors), V22.0201-001
Fall 2008 Semester
Professor Nathan Hull

Project due December 21, 2008
(Hull granted an extension for me)

Part I: Contents Page

Table of Contents

Part I: Contents Page……2
Table of Contents…….2
List of Files……...3
Credits to Parts of Code from other sources…….3
List of Program Modules……..4
Part II: Project Overview……..5
Part III: User Instructions…….6
How to Install Game…….6
Executing the Game…….7
Hints and Features to Know About…….8
Part IV: High Level Technical Details…….9
Basic Description of Graphics…….9
Keyboard Uses……9
Summary of Sounds used in the Game……9
Summary of color codes used for the Game……10
Use of Randomness…….10
Use of the Timer……..11
Brief Summary of all the Modules……..11
The Rectangle Module (RECT)……..13
The Main Playing Area (INNER)……13
Horizontal Line Module (LINEH)……..13
Vertical Line Module (LINEV)……..13
The Bar that Determines Speed……13
The Ball as a Graphic Object (PBALL)……..14
The Obstacles as Graphic Objects (OBS)……14
The Placement of the Obstacles…….14
The Placement of the Ball for Launching……15
Initial Direction of Ball Launch (DIREC)……..16
Animating the Ball Movement……16

Detecting that an Obstacle was hit (OCHECK)……...17
Keeping the Score and Showing It (MESS2) ……17
Part V: Conclusion……18
Appendix I: Original Proposal……..19
Modifications……20
Appendix II: Original Sources…….21

List of Files

DESTEFANOZ.asm
random2.com
das.com
ddb.com
intro.txt
intro2.txt
intro3.txt
Report.doc
Report.txt

Credits to Parts of Code from other sources

The ASCII text art in the introduction was generated by the website:
http://patorjk.com/software/taag/

The parts of the code that writes pixels to the screen were heavily borrowed from the code that writes a single pixel to the screen in the website:
http://www.faqs.org/faqs/assembly-language/x86/general/part1/section-12.html

The module INTRO, PLAYIN, and DOPT was mostly written by fellow student Fatima Rivera

The module PAUSE1, NOTE, and RCTR are originally from the Dewar Game Program

The module DISP is originally from my Sieve of Eratosthenes program earlier in the semester

List of Program Modules (in order of appearance in code)

SDIFF
SETVARS
LAUNCH
MESS1
MESS2
DISP
RANGE
MOVEC
ARROW
ARROW1
ARROW2
ARROW3
ARROW4
DIREC
INTRO
PLAYIN
DOPT
OBSA
OBSB
OBS1
OBS2
NEWXY
OBS3
GETCOL
OBS
ADDT
ADDB
ADDL
ADDR
NOBAR
NEWSCO
CHECKT
CHECKB
CHECKL
CHECKR
BALL
OCHECK
NEGX1
NEGY1
DECX
DECY
PBALL
BAR
FILLB
EMPTYB
INNER
LINEH
RECT
LINEV
PAUSE1
NOTE
RCTR

Part II: Project Overview

The premise of this game is that you are bouncing a bouncy ball around a room with the goal of hitting as many surfaces as possible and not breaking anything in the room. The playing area is a rectangular playing area that simulates a cross-section of a room. There are good and bad obstacles. The good obstacles are colored green and the bad ones are colored red. If the ball hits a bad obstacle, you lose 1 point. If it hits a good obstacle, you gain 2 points. If it hit a wall, you gain 1 point.

The object is to gain as many points as possible in the round. Every time something is hit, the ball slows down until the point that the ball stops at which point the round ends. Every time an obstacle is hit, the obstacle disappears.

The game starts and you are given a bar that determines the speed. The higher the bar goes, the faster that the ball will be launched at. Then you decide where you want the ball to launch from. You are given the acceptable range of places that it can be launched from. You then decide between 4 different angles to launch the ball at: 45 degrees, 135 degrees, -45 degrees, or -135 degrees. The ball then launches and goes around the room until it stops. You then continue to the next round to gain more points. Each game, you are allowed to have 3 rounds to gain as many points as possible.

One of the most interesting achievements was being able to randomly generate the placement of the obstacles and the state of the obstacles so that it would always keep the user guessing and they could never completely figure it out. According to the method of the programming code, there are 460 different rooms possible.

Part III: User Instructions

How to Install game

System Requirements

- Any computer that is capable of running DOS programs
- If you are not running Windows XP, you need to download and install DOSBox and run the program using DOSBox

Installation Instructions

1. Put ALL of the files in the .zip file into a single folder in the directory
    - IMPORTANT: make sure there are no blank spaces in the directory name. Use "_" to signify a blank space
2. If you are running Windows XP or lower, open the command prompt using the following instructions and skip to step 4
    - Click "Start", then "Run", then type in "cmd" and Press Enter.
3. If you are running Windows Vista or another operating system in which this program does not run on, you need to install DOSBox
    - Download DOSBox from http://www.dosbox.com/
    - Install it using their instructions
4. Then set the directory to be the directory where you stored the files. Let "[directory]" represent the directory where the files are stored
    - If you are using the command prompt, type in "cd [directory]"
    - If you are using DOSBox, then type in "mount c [directory]" and press Enter. Then type in "c:" and press enter
5. Type in "random2" and Press Enter to install the random number generator
    - IMPORTANT: must be done every time the command prompt or DOSBox is opened up to run this program
6. Type in "das DESTEFANOZ" and Press Enter to install the program
7. It is now installed and you can type "DESTEFANOZ" and Press Enter to play the game.

Executing the Game

1. Each type you execute the game, these steps have to be completed before you can play the game
    a. Open DOSBox or Command Prompt
    b. Set the directory to be the directory where the files are using the instructions in the installation section
    c. Type in “Random2” and Press Enter. You MUST do this or else the program will not run and DOSBox or Command Prompt will crash.
2. Now, when you type “DESTEFANOZ” and Press Enter, you will start the game
3. You are then presented with an introduction screen and 3 options
    a. Press Enter to Continue with the game
    b. Press I to get the instructions on how to play
    c. Press E to exit the game
    d. Note: None of these instructions are case-sensitive
4. When you continue with the game, you first decide the difficulty level. There are 3 options: easy, medium, or hard. They differ by the probability that an obstacle appearing will be a bad obstacle. Press 1 (easy), 2 (medium), or 3 (hard) to decide difficulty. If you Press Enter, the previously set difficulty will be used.
5. Then, the game starts. In the first screen of the game, the bar tells you how fast the ball will move when launched. Press Enter to set the speed here.
6. In the next part, you launch the ball.
    a. First, use the left, right, up and down arrow keys to determine the launching point. The pink rectangle represents the area where you can launch the ball from. Press Enter when you have decided
7. Then, you decide the angle it will be launched at. The line points in the direction that it will be launched at. Press W to rotate the line counter-clockwise and S to rotate the line clockwise. Press Enter and the Ball will be launched
8. While the ball is moving, press E to exit the game. After it has stopped, you are given the score the score so far and if you Press Enter, the next round will occur. If you press E, you will exit the game.
9. After three rounds, you can decide to exit the game or start a new game.

Hints and Features to Know About

- Try and make the ball launch as fast as possible in order to insure that it has the longest time to bounce around.
- It would be too much of a give-away if you were able to point the ball at a good obstacle when you were launching it and have it hit that obstacle. Therefore, oftentimes, the ball will not detect the obstacle that it hits if it hits an obstacle when it is first launched.
- Due to a minor glitch, in a few rare cases, an obstacle won't be detected to be hit. It rarely ever happens and when it does, it has an unnoticeable effect on your score.

Part IV: High Level Technical Details

Basic Description of Graphics

- Graphics Mode 13h in interrupt 10h was used for the game graphics
    - In this mode, the DI register was used to carry the pixel address and "stosb" was used to write a pixel to the screen.
    - In this graphics mode, the top left most point is considered (0,0) in terms of (x,y) coordinates and the bottom right most point is (320,200). The DI pixel address is determined by the equation DI = 320*y + x
    - The AL register carried the information on the color of the pixel
- Graphics mode 2h in interrupt 10h was used to display the introductory text, instructions, difficulty level information, and score information.
- Interrupt 21h was used to show files and a line of text
    - sub-function 3Dh was used to show a file
    - sub-function 09h was used to show a line of text

Keyboard Uses

- Interrupt 16h was used to get input from the keyboard
- In some places, sub-function 0 was used when the program wanted input from the user.
- In other places, sub-function 1 was used in case the user wanted to do exit during the running or to see when they typed in a key.

Summary of the Sounds used in the game

- Notes with duration 7 circa the Dewar Game Program are used at various points. They have different frequencies depending on what happened.
- During the setup, some sounds are made
    - Once user decides speed: Frequency of 250 Hz used
    - When user has decided spot of ball launch: Frequency of 500 Hz
    - When direction is decided and ball is launched: 750 Hz frequency
- During animation, some sounds are made
    - When a bad obstacle is hit: 1000 Hz frequency

- When wall is hit during animation: 1250 Hz frequency
- When a good obstacle is hit: 1500 Hz frequency

Summary of the Color Codes for the Game

Light Blue (used for main playing area): 036h

Army Green (used for the bar): 090h

Orange (used for the ball): 02Ah

Pink (used to show range where ball can be launched from): 025h

Bright Green (used for line showing direction): 030h

Green (used for the good obstacles): 02Fh

Red (used for the bad obstacles): 070h

Use of Randomness

- The Random Number Generator is used for four things in this game that are related to the obstacles
- It is used to determine if an obstacle will be present. It randomly generates a number from 0-9 inclusive and if the number is 0-3, then an obstacle is not drawn. Therefore, an obstacle has a 60% probability of appearing.
- Then it generates a random number from 0-10 inclusive to determine the offset of the obstacle in the interval that it is allowed to be in. More on placement of the obstacles in the module section of the report.
- For the corners, it also determines if the obstacle will be offset on the x side or the y side. It just generates 0 or 1 to decide the side so each side has a 50% change of the obstacle being there.
- Then it generates a number from 0-9 and based on the difficulty set, decides if the obstacle will be good or bad. If set to easy, then 5 or below means bad one. If set to medium, then 7 or below gets a bad one. If set to hard, then 9 or below gets a bad one. Therefore, the respective probabilities of the obstacle being bad are 50%, 70%, or 90%.

Use of the Timer

- The timer is used for the animation in certain parts of the program. It is from the Dewar Game Program
- For the bar, there is a pause of 0.01 secs between the drawing of each vertical line that fills in the bar or depletes it.
- For the ball, there is a pause of 0.04 secs between the drawing of each iteration of the ball during its animation.

Brief Summary of all the Modules (in order of appearance in code)

Module name	**Purpose of Module**
SDIFF	gets the difficulty level from the user
SETVARS	initializes the variables that reads the placement of obstacles
LAUNCH	sets the (x,y) location of the ball when it first launches
MESS1	displays “set speed” when the bar fills up and empties
MESS2	displays the score in between rounds
DISP	displays the decimal digits of the score
RANGE	displays the area where the ball can be launched from
MOVEC	method that lets user decide where to launch the ball from
ARROW	calls methods that displays the arrow that shows direction
ARROW1	makes arrow that points in 45 degree direction
ARROW2	makes arrow that points in 135 degree direction
ARROW3	makes arrow that points in -135 degree direction
ARROW4	makes arrow that points in -45 degree direction
DIREC	lets user decide the direction to point the ball in
INTRO	prints the introduction file, first thing that the user sees
PLAYIN	prints the instructions on how to play if user requests it
DOPT	displays the options to user for difficulty settings
OBSA	sets up methods that make obstacles in top/bottom of screen
OBSB	sets up methods that make obstacles in left/right of screen
OBS1	writes multiple obstacles in the top or bottom of screen
OBS2	writes multiple obstacles in the left or right of screen

NEWXY	randomly offsets the x or y coordinate of the corner obstacles
OBS3	writes multiple obstacles in the corners of the screen
GETCOL	randomly decides the color of the obstacle
OBS	writes a single obstacle
ADDT	adds obstacle information to the arrays about the top obstacles
ADDB	adds obstacle information to the arrays about the bottom obstacles
ADDL	adds obstacle information to the arrays about the left obstacles
ADDR	adds obstacle information to the arrays about the right obstacles
NOBAR	paints over the bar after user has decided the speed
NEWSCO	updates the score if an obstacle was hit by the ball
CHECKT	checks to see if ball has hit an obstacle on the top
CHECKB	checks to see if ball has hit an obstacle on the bottom
CHECKL	checks to see if ball has hit an obstacle on the left side
CHECKR	checks to see if ball has hit an obstacle on the right side
BALL	animates the ball moving around the room
OCHECK	checks to see if ball has hit an obstacle
NEGX1	called if ball hits left or right wall. it changes direction of ball
NEGY1	called if ball hits top or bottom wall. it changes direction of ball
DECX	slows down the ball by decreasing the change in x
DECY	slows down the ball by decreasing the change in y
PBALL	used to display the ball. called by the method that animates the ball
BAR	prints the outline of the bar that determines the speed
FILLB	this animates the bar filling in
EMPTYB	this animates the bar emptying
INNER	prints the rectangle that shows the main playing area
LINEH	prints a horizontal line
RECT	prints a rectangle
LINEV	prints a vertical line
PAUSE1	pauses the animation
NOTE	plays a note (from the Dewar Game Program)
RCTR	helps note program run (from Dewar Game Program)

The Rectangle Module (RECT)

- The module is called multiple times to draw a rectangle. It uses the COLOR variable to determine the color of the rectangle. The variables XMIN, YMIN, XLEN, and YLEN are used to draw the rectangle.
- XMIN and YMIN are coordinates of top left corner. XLEN is horizontal length and YLEN is vertical length of rectangle.

The Main Playing Area (INNER)

- The default background is used to show the border. The main playing area is a light blue rectangle that is drawn using the Rectangle Module.
- XMIN: 5; YMIN: 5; XLEN: 310, YLEN: 190
- This creates a 5-pixel border all around the playing area that represents a wall

Horizontal Line Module (LINEH)

- Used by a few modules to draw a horizontal line.
- (XMIN, YMIN) determine the coordinates of the left most point
- XLEN determines the length of the line

Vertical Line Module (LINEV)

- Used by a few modules to draw a vertical line.
- (XMIN, YMIN) determine the coordinates of the top most point
- YLEN determines the length of the line

The Bar that Determines Speed

- BAR, FILLB, EMPTYB modules used
- The outline is drawn using LINEH and LINEV by the BAR module
    - The left line is at x=80. The right line is at x=240.
    - The top line is at y=145. The bottom line is at y=155
- The FILLB method fills in the bar by animating the drawing of a series of vertical lines making a rectangle. If the user types a key, it stops and sets the speed there.

- The EMPTYB method depletes the bar after it was filled in using the same method just with background colored lines so it appears to be depleting. It is only called if the user did not type a key yet. When the user types a key, it stops and the speed is set

The Ball as a Graphic Object (PBALL)

- It is rectangle where the XMIN and YMIN coordinates are determined by the variables BALLX and BALLY respectively. Those two variables are determined by the module that calls this module.
- It is a 3 pixel by 3 pixel square that is colored orange.
- The rectangle module is called using these parameters to draw the ball.

The Obstacles as Graphic Objects (OBS)

- Whether they appear or not is determined by this module that uses the random number generator to decide that.
- They are 10 pixel by 10 pixel rectangles that are drawn using the rectangle module. The XMIN and YMIN for the rectangle are determined by the variables OBSX and OBSY which are set by the modules that call this one.
- The color is either green or red, depending on what the random number generator decides.

The Placement of the Obstacles

- Introduction
    - The sides of the playing area are broken into intervals of 40 pixels each. The corners (end intervals) are dealt with by a special method.
    - The other intervals are dealt with relatively the same way
    - The OBSA method calls OBS1 twice with two different OBSY values to determine the placement of the top and bottom obstacles.
    - The OBSB method calls OBS2 twice with two different OBSX values to determine the placement of the left and right obstacles

    - The OBS3 method determines the placement of obstacles that are in the corners.
- Summary for middle of side (OBS1, OBS2)
    - Each interval has 4 different parts. The first 10 and last 10 pixels are dead zones where no obstacle will appear
    - Let's refer to the beginning of the interval as "I"
    - From point (I + 10, I + 20) is where the beginning of the obstacle could be placed.
    - The random number generator decides where in this interval the obstacle will be placed. Remember that the obstacle will take up 10 pixels, so only that narrow window is tolerable.
    - It then draws the obstacle using the OBS module
- Summary for the corner obstacles (OBS3)
    - There can only be 1 obstacle per corner. In this case, it is only drawn in the first 10 pixels of each side.
    - The random number generator decides whether it will be on x or y side.
    - Then, the random number generator decides the x or y offset for the corner obstacle.
    - It is then drawn using the OBS module
- Total possible arrangements for the obstacles.
    - There are 20 possible obstacles that can appear (4 corner ones, 6 on top/bottom, and 4 on left/right)
    - Each obstacle has 23 possible arrangements (1 is that it does not appear, then 11 places for a good obstacle, or 11 places for a bad obstacle)
    - Multiply these together and there are 460 arrangements.

<u>The Placement of the Ball for Launching</u>

- The modules used are LAUNCH, RANGE, and MOVEC
- LAUNCH runs this operation. It calls RANGE to show the range where the ball can be launched. The range shown is a hollow rectangle drawn with horizontal

and vertical lines. The left side is at x = 16. The right side is at x = 185. The top side is at y=16. The bottom side is at y=304.

- MOVEC is then called which moves the ball up and down or left and right to determine it's launching point. It uses PBALL to paint the ball at each point and the keyboard gets the input from the user as to what direction to move it.

Initial Direction of Ball Launch (DIREC)

- This shows the line that determines the direction. It uses the keyboard to get input on whether to move the line clockwise or counter-clockwise. This manipulates a variable called DIR that is used by the ARROW method to determine what arrow to draw.
- The Arrow module calls ARROW1, ARROW2, ARROW3, or ARROW4 depending on what direction the line needs to be drawn in
- Each of those modules draws the line that shows direction.
- When enter is pressed, the ball is launched in that direction.

Animating the Ball Movement

- With each iteration, it paints over its current place and paints the ball at the next place. It calls the PBALL module to paint the ball.
- If the top or bottom wall is hit, it calls the NEGY1 module to negate the change in y during animation. This module also increments the score and decrements the velocity of the ball.
- If the left or right wall is hit, it calls the NEGX1 module to negate the change in x during animation. This module also increments the score and decrements the velocity of the ball.
- It calls the PAUSE1 module with a delay of 0.04 seconds in order to show the steps to the user so that it does not animate too fast.
- It then calls the OCHECK module to see if an obstacle was hit.
- It then checks to see if the change in x or y would be zero in the next iteration and stops the animation if that would be true, because then the ball would have lost all momentum.

Detecting that an Obstacle was hit (OCHECK)

- It first checks to see if the ball is in the range of a set of obstacles.
    - If the y coordinate is below 15, it could be hitting a top obstacle. In this case, the module CHECKT is called which checks the top.
    - If y coordinate is above 185, it could be hitting a bottom obstacle. In this case, the module CHECKB is called which checks the bottom.
    - If the x coordinate is below 12, it could be hitting a left obstacle. In this case, the module CHECKL is called which checks the left side.
    - If x coordinate is above 302, it could be hitting a right obstacle. In this case, the module CHECKR is called which checks the right side.
- Each of the modules go through their respective arrays which have the information on the placement of the obstacles and sees if the ball is in the range of any of the obstacles.
    - Let's set C to be the x/y coordinate of the ball depending on what is being tested. If top or bottom is tested, then C is the x coordinate. If left or right is tested, then C is the y coordinate.
    - Let T be the x/y coordinate of the obstacle depending on what is being tested. If top or bottom is tested, then T is the x coordinate. If left or right is tested, then T is the y coordinate.
    - If C is in the range of [T-3, T+10], then the obstacle has been hit, so the module registers that.
    - If an obstacle is hit, it paints over the obstacle, erases its information in the array, and then sees if it is green or red using another array and calls the NEWSCO module to update the score accordingly.

Keeping Score and showing it (MESS2)

- The MESS2 module is called between rounds to show the score. The score variable is changed by the NEGX1, NEGY1, and NEWSCO module as described above.
- The module uses the DISP module to show the decimal digits of the score.

Part V: Conclusion

After everything was said and done in this program, including basic and advanced features that I wanted to include, the final count of the length of the program including the comments was 1600 lines. It took an extremely long time to write all of this and make it work perfectly. I must have spent around 50 hours programming this game. This game proved to be very difficult to program and there is not a lot that I could have done differently. I only hope that I knew about DOSBox sooner, because it severely delayed me not being able to use the 13h graphics mode in the Windows Vista Command Prompt.

It was a good experience programming this and it taught me all about how some graphics are programmed. I also gained an appreciation for how difficult it is to program video games. You never realize while playing them how much code has to go into making it work. Once you have to program this, you fully realize it. My game looks like something from 20 to 30 years ago, but even it took a very long time and a lot of effort to program. If I had more time, I would have figured out how to write data to the hard drive so that my game could keep a list of high scores and the users who scored the highest.

Appendix I: Original Proposal

The idea of this game is that you are launching a bouncy ball around a room and trying to get it to hit as many walls and good objects as possible and trying to have it avoid bad objects. The game starts off and the user is presented with a large rectangle that is a 2-D version of a room, which we can think of as a cross-section of a room. The top and bottom edge is the ceiling and floor respectively and the side edge is the wall. On each edge in random spots are "good" obstacles, which give the user a point if he hits them and the "bad" obstacles which takes away points from the user if they get it.

The user is then presented with a "launcher" which will look slightly like a canon. It is put into the center of the room. The user rotates the cannon around to get the angle that they prefer. Then, they have to decide the speed to launch it at. A bar moves forward and the higher that the bar goes, the faster the ball will launch at. When the bar goes to its maximum value, it will go down. When the user presses a button, the speed of the ball launch will be the speed that the bar's location signifies. Then, they are given an accuracy which works the same way, except that the accuracy bar represents the interval at which the angle can be offset by. In the center of the bar is 0, where it is perfectly accurate. The beginning and end of the bar represents plus or minus 20 degrees. The accuracy bar will move faster if the player chose a higher speed. Once accuracy is decided on, the ball launches at an angle that is the chosen angle plus or minus the maximum offset.

The ball then bounces around the room. For every edge or good obstacle hit, the user gets 1 point. For every bad obstacle hit, the user is deducted one point. The program will simulate when the bouncy ball loses momentum, at which point, the round is over. The user will then get two more rounds to increase his score.

I will use VGA graphics to implement this because multiple objects and various angles would look awkward in lower level graphics. A random number generator will be used to generate the placement of the good and bad obstacles. I will use the keyboard buttons to have the user set the angle and stop on the speed and accuracy bar. The user friendly features will include a short description of the game before it starts and the user will press the enter key when they have finished reading the description. At the bottom will be a short description of what the user is doing at each stage. There will be a short line describing that the user is setting the angle, setting the speed, and setting the

accuracy. The user will also have the option of setting the level of the game. The higher the level, the more obstacles there are on screen.

The basic features of the game include the launching of the bar, pre-made rooms with good and bad obstacles, the bouncing of the ball around the room, and the score keeper. The more advanced features will include setting the level, customizable rooms, randomly generating rooms. If I have time to go beyond this, I will have a progressively harder game where after the user completes his rounds in one room, he moves to a slightly harder room if his score is high enough. Eventually, he will beat the game when he completes the hardest level.

Modifications

- Instead of a canon in the center launching the ball, the user decides where to launch it and has only 4 directions to launch it in.
- The instructions on how to play the game at each step are displayed if the user requests it and not displayed during the execution of the program, except for setting the speed using the bar.
- The obstacles disappear when hit.
- Instead of levels, the user just decides the difficulty.

Appendix II: Original Sources

Pixel Module (retooled in various parts of the program)

```
;from website: http://www.faqs.org/faqs/assembly-language/x86/general/part1/section-
;12.html
PIXEL PROC
      ;prepare to write to video buffer
         mov    di, 0a000h      ;put the video segment into di
         mov    es, di          ;so it can be easily put into ds
         xor    di, di          ;start writing at coordinates (0,0)
         mov    ax, 320
         mul    Y
         mov    di, ax
         add    di, X
         mov    al, COLOR
         stosb                  ;writes the pixel to the screen
         ret
PIXEL ENDP
```

Print1.asm (from Fatima Rivera. used in program to print files)

```
OpenFileErrorMessage db 'No such file.','$'
ReadErrorMessage db 'Cannot read from file.','$'
buffer          db      100 dup (?), '$'
file1           db      'file1.TXT',0,8,'$'

start:
print_file proc
         mov    dx, offset file1
         mov    ah, 3dh         ;DOS function to open a file
         int    21h             ;open the file
         jc     openError       ;if opening error, display error
         mov    bx, ax          ;save  file handle to bx because read procedure will need it
printlp:

         mov    ah, 3fh         ;DOS function to read file
         lea    dx, buffer
         mov    cx, 100         ;number of bytes to read
         int    21h

         jc     readError               ; if error

         mov    si, ax                  ;number of bytes read
         mov    buffer[si], '$'         ;need $ to print buffer array

         mov    dx, offset buffer       ;buffer is the array to read
         mov    ah, 09h                 ;DOS function to print file
```

```
        int     21h                     ;print on screen now!

        cmp     si, 100                 ;check amt of data read
        je      printlp         ;not done reading, read more

        jmp stop                        ;done reading, jump to end


openError:

        mov     ah, 09h
        mov     dx, offset OpenFileErrorMessage
        int     21h
        jmp     stop


readError:
        mov     dx, offset ReadErrorMessage
        mov     ah, 09h
        int     21h
print_file endp
stop:

        mov     ax, 4c00h               ;return control to DOS
        int     21h

        hlt
        end
```