```
IDEAL
MODEL small
STACK 100h
DATASEG
    Clock equ es:6Ch
    randomLetter db ? ;saves the random letters generated
    randomColumn db ? ;saves the random line generated
    randomColor db ? ;saves the random color generated
    RandomMemoryPlace db 9 ;random number for the random procedures
    delayTime dw 18 ;delay time is calculated: (time to delay)/0.055. Default delay time is about
    a second.
    blinkerTime dw 5 ;delay time of a quarter of a second
    pressedKey db ? ;saves the keys pressed during the program
    row db 0 ;initialize the first line
    lettersArray db 7 dup (0) ;array of random letters on the screen
    columnArray db 7 dup (0) ;array of random column generated per each random letter
    letterJunk db?
    columnJunk db?
    rowNumber db?
    columnNumber db?
    position db?
    lose db 0 ;lose is either 0 or 1 which is significant for true or false
    -0;false, 1-true
    points dw 5d ;Default points is for Easy mode
    ;points:
```

```
;Easy - 5pts per correct press - default

;Intermediate - 10pts per correct press

;Hard - 15pts per correct press

;Impossible - 20pts per correct press

score dw 0 ;saves the game score

newScore dw 0 ;saves the game score that can be changed without worrying about the regular score

;opening

gameTitle db 10,13,10,13,10,13,'                _____        _              ',10, 13
  db '               |_  _|       (_)             ',10, 13
  db '                | |_  _ _ __ _ _ __  __ _     ',10, 13
  db "                | | | | '_ \| | '_ \ / _` |    ",10, 13
  db '                | | |_| | |_) | | | | (_| |    ',10, 13
  db '                \_/\__, | .__/|_|_| |_|\__, |    ',10, 13
  db '                   __/ | |          __/ |    ',10, 13
  db '                  |___/|_|         |___/     ',10, 13
  db '                   ___  ___          _       ',10, 13
  db '                   |  \/  |         (_)      ',10, 13
  db '                   | .  . | __ _ _ __  _  __ _  ',10, 13
  db "                   | |\/| |/ _` | '_ \| |/ _` |  ",10, 13
  db '                   | |  | | (_| | | | | | (_| |  ',10, 13
  db '                   \_|  |_/\__,_|_| |_|_|\__,_|  ',10, 13, 10, 13, 10, 13, 10, 13
        db '                        ENTER TO START ', 10, 13'$' ,


gameSubTitle db 'Pick a Stage, Press enter to select:     ',10,13,10,13,10,13'$',
```

```
option1     db '                >> Easy         ',10,13,10,13
            db '                   Intermediate    ',10,13,10,13
            db '                   Hard         ',10,13,10,13
            db '                   Impossible     ',10,13,10,13
            db '                   Help         ',10,13,10,13
            db '                   Quit         ',10,13,10,13,10,13'$',

option2     db '                   Easy         ',10,13,10,13
            db '                >> Intermediate   ',10,13,10,13
            db '                   Hard         ',10,13,10,13
            db '                   Impossible     ',10,13,10,13
            db '                   Help         ',10,13,10,13
            db '                   Quit         ',10,13,10,13,10,13'$',

option3     db '                   Easy         ',10,13,10,13
            db '                   Intermediate    ',10,13,10,13
            db '                >> Hard         ',10,13,10,13
            db '                   Impossible     ',10,13,10,13
            db '                   Help         ',10,13,10,13
            db '                   Quit         ',10,13,10,13,10,13'$',

option4     db '                   Easy         ',10,13,10,13
            db '                   Intermediate    ',10,13,10,13
```

```
        db '                    Hard          ',10,13,10,13
        db '                  >> Impossible     ',10,13,10,13
        db '                    Help          ',10,13,10,13
        db '                    Quit          ',10,13,10,13,10,13'$',

    option5         db '                      Easy          ',10,13,10,13
        db '                    Intermediate    ',10,13,10,13
        db '                    Hard          ',10,13,10,13
        db '                    Impossible      ',10,13,10,13
        db '                  >> Help          ',10,13,10,13
        db '                    Quit          ',10,13,10,13,10,13'$',

    option6         db '                      Easy          ',10,13,10,13
        db '                    Intermediate    ',10,13,10,13
        db '                    Hard          ',10,13,10,13
        db '                    Impossible      ',10,13,10,13
        db '                    Help          ',10,13,10,13
        db '                  >> Quit          ',10,13,10,13,10,13'$',

    credits      db '                   Special thanks to:          ',10,13
        db '                  * Amit Keinan *           ',10,13
        db '                  * Alon Sarel *           ',10,13
        db '                  * Ron Yutkin *           ',10,13,10,13
        db '              ',175,' MADE BY YUVAL STEIN, ISRAEL ',174,10,13 ;175 and 174 are ASCII for
'»' '«'
```

```
 db '                    Version 1.0.1$'
;mode
- 1;    Easy - default
- 2;    Intermediate
 - 3;Hard
 - 4;Impossible
 - 5;Help -> Display instructions
 - 6;Quit
mode db 1

 instructions db      'Welcome to typing mania!',10,13
 db       'In this game your goal is to pop as many letters as you can!',10,13
 db '   In order to pop a letter you must type the letter you see.',10,13
 db '   Pay Attention! You can only pop the lowest letter on the screen...',10,13
 db '   The points you will get are calculated according to the level you chose: ',10,13
  db '   Easy - 5pts per each letter ',10,13
 db '    Intermediate - 10pts per each letter ',10,13
 db '    Hard - 15pts per each letter ',10,13
 db '    Impossible - 20pts per each letter...if you can pop any...',10,13
 db '  GOOD LUCK! ',1,' ',2,' ',1,' ',2,' ',1,10,13,'$' ;the numbers 1&2 are ASCII of a smiley faces
(☻-2 ,☺-1)
opt1          db 10, 13, 10, 13, 10, 13, '                    >    Got it'!
 db           10, 13, '                       Quit'$

opt2          db 10, 13, 10, 13, 10, 13, '                        Got it'!
```

```
 db            10, 13, '                    >     Quit'$
optState     db 1 ;optState is either 1 or 2 (1-go to main menu, 2-quit)

dispScore    db '_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _   ',10,13
 db '              |                        |   ',10,13
 db '              |          Score:              |   ',10,13
 db '              |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|   ',10,13,10,13,10,13,10,13'$',

op1                db    '                    <       NewGame                ',10,13,10,13
 db    '                          Quit                ',10,13 '$',

op2                db    '                          NewGame                ',10,13,10,13
 db    '                    <       Quit                ',10,13 '$',

opState            db 1 ;opState is either 1 or 2 (1-NewGame -> go to main menu, 2-quit)

cursorPosition db 44d ;last column after word 'Score: ' for last digit in [score] - in order to print the score in the right place
digit db?


CODESEG


proc ClearScreen
;the procedure clears the screen by going into text mode
push ax
```

```
    mov al, 03h ;text mode (80x25)
    mov ah, 0 ;set video mode
    int 10h

    pop ax
    ret
endp ClearScreen

proc PrintMainMenu
    ;Procedure prints main Menu
    push ax
    push cx
    push dx

    call ClearScreen
    mov dx, offset gameTitle
    mov ah, 09
    int 21h
    ;hide cursor
    mov cx, 2607h
    mov ah, 1
    int 10h
WaitForEnter:
```

```
    ;Check if a key was clicked
    mov ah, 1
    int 16h
    jz WaitForEnter
    mov ah, 0
    int 16h
    cmp ah, 1h ;if esc clicked Quit
    je Quit
    cmp ah, 1Ch ;check if enter key is pressed according to the scan code
    jne WaitForEnter
    jmp Dontexit
Quit:
    mov ax, 4c00h
    int 21h
Dontexit:
    pop dx
    pop cx
    pop ax
    ret
endp PrintMainMenu

proc PrintGameOptionsMenu
    ;Procedure prints the second page of the game - Options Menu
    push ax
```

```
push bx
push dx

call ClearScreen

;Move cursor to a specific location using BIOS Interrupt 10h
mov ah, 2h
mov bh, 0h ;page 0
mov dh, 2h ;row
mov dl, 20d ;column
int 10h
;Print Game Options
mov dx, offset gameSubTitle
mov ah, 09
int 21h
mov dx, offset option1
mov ah, 09
int 21h

;Print Credits
mov dx, offset credits
mov ah, 09
int 21h
;hide cursor
```

```
    mov cx, 2607h
    mov ah, 1
    int 10h

WaitForKeyPress:
    mov ah, 1
    int 16h
    jz WaitForKeyPress
    mov ah, 0
    int 16h
    cmp ah, 1h ;if esc pressed quit
    je ExitGame
    cmp ah, 1Ch ;if enter is pressed take action
    je TakeActionAccordingly
    cmp ah, 50h ;if down key is pressed
    jne CheckUpKey
    ;change option if the option isn't bigger than the last option possible
    cmp [mode], 6
    je DontIncMode
    inc [mode]
DontIncMode:
    jmp UpdateScreenStatus
CheckUpKey:
    cmp ah, 48h ;if up key is pressed
```

```
    jne WaitForKeyPress ;if none of these buttons were clicked wait for a good click (either enter, up or down)
    ;change option if the option isn't smaller than the first option
    cmp [mode], 1
    je UpdateScreenStatus
    dec [mode]
    jmp UpdateScreenStatus
TakeActionAccordingly:
    jmp TakeFurtherAction
ExitGame:
    mov ax, 4c00h
    int 21h
UpdateScreenStatus:
    call ClearScreen ;clear the present screen and type it again updated
    ;Move cursor to the same spot were it was before
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 2h ;row
    mov dl, 20d ;column
    int 10h
    ;Print Game Options
    mov dx, offset gameSubTitle
    mov ah, 09
    int 21h
    ;Change option if nessesary
```

```
    cmp [mode], 2
   je Mode2State
    cmp [mode], 3
    je Mode3State
    cmp [mode], 4
   je Mode4State
    cmp [mode], 5
    je Mode5State
    cmp [mode], 6
    je Mode6State
Mode1State:
    mov dx, offset option1
    jmp pri
Mode2State:
    mov dx, offset option2
    jmp pri
Mode3State:
    mov dx, offset option3
    jmp pri
Mode4State:
    mov dx, offset option4
    jmp pri
Mode5State:
    mov dx, offset option5
```

```
    jmp pri
Mode6State:
    mov dx, offset option6
pri:
    mov ah, 09
    int 21h
    ;Print Credits
    mov dx, offset credits
    mov ah, 09
    int 21h
    ;hide cursor
    mov cx, 2607h
    mov ah, 1
    int 10h
    jmp WaitForKeyPress
TakeFurtherAction:
    call TakeActionAccordingToMode
    cmp [mode], 7 ;if instructions were printed go back to WaitForKeyPress in order to get new mode
    jne InstructionsWereNotPrinted
    mov [mode], 1 ;reset mode to default
    jmp UpdateScreenStatus
InstructionsWereNotPrinted:
    pop dx
    pop bx
```

```
    pop ax
    ret
endp PrintGameOptionsMenu


proc TakeActionAccordingToMode
    ;Procedure initializes the delayTime according to the mode chosen
    cmp [mode], 6 ;option 6 is Quit
    je QuitGame
    jmp Contin
QuitGame:
    mov ax, 4c00h
    int 21h
Contin:
    ;Check the option pressed and update the required things for the game according to the mode
    ;mode 1 is default and therefore doesn't need to be checked specialy
    cmp [mode], 5 ;if mode = 5 display instructions
    je DisplayIstructionsForHelp
    jmp CheckNextMode
DisplayIstructionsForHelp:
    call PrintInstructions
CheckNextMode:
    cmp [mode], 7 ;if instructions were printed go back to menu in order to get new mode
    je ReturnToMenu
    cmp [mode], 2 ;if mode = 2 - Intermediate level
```

```
    je ChangeVarsToIntermediate
    cmp [mode], 3 ;if mode = 3 - Hard level
    je ChangeVarsToHard
    cmp [mode], 4 ;if mode = 4 - Impossible level
    je ChangeVarsToImpossible
    jmp ReturnToMenu ;if mode = 1 - Easy level, continue with default delay time and default points
ChangeVarsToIntermediate:
    mov [delayTime], 10 ;is equal to about half a second
    mov [points], 10d ;10pts for each correct press in Intermediate mode
    jmp ReturnToMenu
ChangeVarsToHard:
    mov [delayTime], 6 ;is equal to about a quarter of a second
    mov [points], 15d ;15pts for each correct press in Hard mode
    jmp ReturnToMenu
ChangeVarsToImpossible:
    mov [delayTime], 1 ;Almost no delay at all - nearly impossible
    mov [points], 20d ;20pts for each correct press in Impossible mode
ReturnToMenu:
    ret
endp TakeActionAccordingToMode


proc PrintInstructions
    ;Procedure prints instructions if requested (if mode is 5)
    push ax
```

```
push bx
push dx

call ClearScreen

;Move cursor to a specific location using BIOS Interrupt 10h
mov ah, 2h
mov bh, 0h ;page 0
mov dh, 2h ;row
mov dl, 5h ;column
int 10h
;Print Instructions
mov dx, offset instructions
mov ah, 09
int 21h

;Print Options
mov dx, offset opt1
mov ah, 09
int 21h
;hide cursor
mov cx, 2607h
mov ah, 1
int 10h
```

```
    ;Check if down key or up key is pressed and take action accordingly
WaitForKey:
    mov ah, 1
    int 16h
    jz WaitForKey
    mov ah, 0
    int 16h
    cmp ah, 1h ;if esc pressed quit
    je ExitBeforeGame
    cmp ah, 1Ch ;if enter is pressed take action
    je TakeAction
    cmp ah, 50h ;if down key is pressed
    jne CheckUp
    ;change option
    mov [optState], 2
    jmp UpdateScreen
CheckUp:
    cmp ah, 48h ;if up key is pressed
    jne WaitForKey ;if none of these buttons were clicked wait for a good click (either enter, up or down)
    ;change option
    mov [optState], 1
UpdateScreen:
    call ClearScreen ;clear the present screen and type it again updated
    ;Move cursor to the same spot were it was before
```

```
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 2h ;row
    mov dl, 5h ;column
    int 10h
    ;Print Instructions
    mov dx, offset instructions
    mov ah, 09
    int 21h

    ;Change option if nessesary
    cmp [optState], 2
   je opt2State
opt1State:
    mov dx, offset opt1
    jmp pr
opt2State:
    mov dx, offset opt2
pr:
    mov ah, 09
    int 21h
    ;hide cursor
    mov cx, 2607h
    mov ah, 1
```

```
    int 10h
    jmp WaitForKey
TakeAction:
    cmp [optState], 2
    je ExitBeforeGame
    jmp Continu
ExitBeforeGame:
    mov ax, 4c00h
    int 21h
Continu:
    pop dx
    pop bx
    pop ax
    mov [mode], 7 ;mode that does not exist, for checking if instructions were already displayed or not
    ret
endp PrintInstructions


proc GetRandomLetter
    ;Procedure generates a random letter and saves it


NewRandomLetter:
    mul bx
    ;Get random number between 0-31
    mov ax, 40h
```

```
    mov es, ax
    mov ax, [Clock] ;read timer counter
    mov ah, [byte cs:bx] ;read one byte from memory
    xor al, ah ;xor memory and counter
    and al, 00011111b ;leave result between 0-31

    ;Take number if between 0-25 - only 26 letters in the English alfhabet
    cmp al, 26
    jae NewRandomLetter
    add al, 'a' ;ascii number of the letter
    mov [randomLetter], al

    ret
endp GetRandomLetter

proc GetrandomColumn
    ;Procedure generates a random column and saves it

NewRandomColumn:
    sub bx, 3 ;for smooth running (faster good random number generated here)
    ;Get random column between 0-79
    mov ax, 40h
    mov es, ax
    mov ax, [Clock] ;read timer counter
```

```
    mov ah, [byte cs:bx] ;read one byte from memory
    xor al, ah ;xor memory and counter
    and al, 01111111b ;leave result between 0-127
    ;if number is above 79 it will be printed in this column anyway

    ;Take number if it isn't on the borders (0 or 79)
    cmp al, 0
    je NewRandomColumn
    cmp al, 79
    jae NewRandomColumn
    mov [randomColumn], al

    ret
endp GetrandomColumn

proc GetRandomColor
    ;Procedure generates a random color and saves it

NewRandomColor:
    add bx, 7
    ;Get random number between 0-16
    mov ax, 40h
    mov es, ax
    mov ax, [Clock] ;read timer counter
```

```
    mov ah, [byte cs:bx] ;read one byte from memory
    xor al, ah ;xor memory and counter
    and al, 00001111b ;leave result between 0-16

    ;Take number if it isn't 0 because 0 is the color attribute of black and
    ;the background color is black, therefore the letter wont be seen.
    cmp al, 0
    je NewRandomColor
    mov [randomColor], al

    ret
endp GetRandomColor

proc PrintLetter
    ;Procedure gets a random color, letter and column and prints it
    color equ [bp+8]
    letter equ [bp+6]
    column equ [bp+4]

    push bp
    mov bp, sp

    push ax
    push bx
```

```
push cx
push dx

;hide cursor
mov cx, 2607h
mov ah, 1
int 10h

;Move cursor to a specific location using BIOS Interrupt 10h
mov ah, 2h
mov bh, 0h ;page 0
mov dh, 0h ;first row
mov dl, column ;random column
int 10h

;Print letter in a new line each time by scrolling up
mov ah, 7h
mov al, 4h ;number of lines to scroll
mov bh, Color ;color attribute of text and background
mov ch, 0 ;y coordinate of top left
mov cl, 0 ;x coordinate of top left
mov dh, 24d ;y coordinate of lower right
mov dl, 79d ;x coordinate of lower right
int 10h
```

```
    ;Print Random Letter:
    mov al, letter
    mov dl, al
    mov ah, 2
    int 21h

    ;Update current screen status
    mov ax, letter
    push ax
    mov ax, column
    push ax
    call UpdateStatus

    pop dx
    pop cx
    pop bx
    pop ax
    pop bp
    ret 6
endp PrintLetter

proc UpdateStatus
    ;Procedure updates the arrays according to the random letters and columns generated
```

```
    letter equ [bp+6]
    column equ [bp+4]

    push bp
    mov bp, sp

    push ax
    push bx
    push cx

    ;Update letters and their status:
    ;Move the last letter to garbage since it is no longer on the screen
    mov bx, offset lettersArray
    mov al, [bx+6]
    mov [letterJunk], al
    ;Move all the letters in the array one position forward
    mov cx, 6
LoopMoveLettersForward:
    mov [position], cl
    sub [position], 1
    mov bx, offset lettersArray
    add bl, [position]
    mov al, [bx]
    mov [bx+1], al
```

```
    loop LoopMoveLettersForward
    mov ax, letter
    mov bx, offset lettersArray
    mov [bx], al ;store the new random letter generated in the letters array

    ;Move the last column to garbage since it is no longer relevant
    mov bx, offset columnArray
    mov al, [bx+6]
    mov [columnJunk], al
    ;Move all the columns in the column array one position forward
    mov cx, 6
LoopMoveColumnsForward:
    mov [position], cl
    sub [position], 1
    mov bx, offset columnArray
    add bl, [position]
    mov al, [bx]
    mov [bx+1], al
    loop LoopMoveColumnsForward
    mov ax, column
    mov bx, offset columnArray
    mov [bx], al ;store the random column generated for the random letter

    ;check if letter is in last place in the array
```

```
    mov bx, offset lettersArray
    cmp [byte ptr bx+6], 0
    je Con
    mov [lose], 1 ;lose is now true (1-true, 0-false)
Con:
    pop cx
    pop bx
    pop ax
    pop bp
    ret 4
endp UpdateStatus


proc CheckPressedKey
    ;Procedure checks if a _letter_ key is pressed and calls another procedure to erase the pressed letter if nessesary
    key equ [bp+4]

    push bp
    mov bp, sp

    ;Take letter only if it is in the last row on the screen
    mov cx, 7
    ;Find the last letter in the array which is not 0
LoopFindLastLetter:
    mov [position], cl
```

```
    mov bx, offset lettersArray
    add bl, [position]
    dec bx
    cmp [byte ptr bx], 0
    jne Found
    loop LoopFindLastLetter
Found:
    mov ax, key
    mov bx, offset lettersArray
    add bl, [position]
    cmp [bx-1], al ;compare the letter in the array in place [position-1] to the key pressed
    jne DontTakeAction
    call Erase
DontTakeAction:
    pop bp
    ret 2
endp CheckPressedKey

proc Erase
    ;Procedure erases the _letter_ key pressed
    dec [position]
    cmp [position], 6 ;Check if the letter pressed is on the last line on the screen
    je Finish ;if so end game
    mov bx, offset lettersArray
```

```
add bl, [position]
mov [byte ptr bx], 0 ;erase the letter from the array
mov al, 4 ;row is calculated 4*position(in array)
mul [position]
mov [rowNumber], al
mov bx, offset columnArray
add bl, [position]
mov al, [bx]
mov [columnNumber], al
;Move cursor to a specific location using BIOS Interrupt 10h
mov ah, 2h
mov bh, 0h ;page 0
mov dh, [rowNumber] ;row
mov dl, [columnNumber] ;column
int 10h
;Delete letter by writing a blank character(' ')
mov dl' ' ,
mov ah, 2
int 21h
;Add points to score
mov ax, [score]
add ax, [points]
mov [score], ax
jmp ContinueGame
```

```
Finish:
    mov [lose], 1 ;lose is now true (1-true, 0-false)
ContinueGame:
    ret
endp Erase

proc Delay
    ;Procedure gets the time to delay the program and delays it
    timeToDelay equ [bp+4]

    push bp
    mov bp, sp

    push ax
    push cx

    ;wait for first change in timer
    mov ax, 40h
    mov es, ax
    mov ax, [Clock]
FirstTick:
    cmp ax, [Clock]
    je FirstTick
    mov cx, timeToDelay
```

```
DelayLoop:
    mov ax, [Clock]
Tick:
    cmp ax, [Clock]
    je Tick
    loop DelayLoop

    pop cx
    pop ax
    pop bp

    ret 2
endp Delay

proc EndGame
    ;Procedure shows the user where he failed and returns to main screen
    mov bx, offset columnArray
    mov dl, [bx+6] ;column of the last letter on screen

    ;Move cursor to a specific location using BIOS Interrupt 10h
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 24d ;last row
    int 10h
```

```
;change letter to red using the scroll interrupt
;scroll up 1 line
mov ah, 7h
mov al, 1 ;number of lines to scroll
mov bh, 04h ;color attribute of text and background - red text on black background
mov ch, 0 ;y coordinate of top left
mov cl, 0 ;x coordinate of top left
mov dh, 24d ;y coordinate of lower right
mov dl, 79d ;x coordinate of lower right
int 10h
;scroll down 1 line
mov ah, 6h
mov al, 1 ;number of lines to scroll
mov bh, 04h ;color attribute of text and background - red text on black background
mov ch, 0 ;y coordinate of top left
mov cl, 0 ;x coordinate of top left
mov dh, 24d ;y coordinate of lower right
mov dl, 79d ;x coordinate of lower right
int 10h
;Change text mode to 1 in order to toggle intensity/blinking:
mov ax, 1003h
mov bl, 1 ;enable blinking
mov bh, 0
int 10h
```

```
    mov dx, 5 ;dx works as counter because cx is already used in the interrupt
    ;since the delay is 0.25[s] each time the total blinker time is 1.25[s]
BlinkerLoop:
    ;show box-shaped blinking text cursor:
    mov ch, 0
    mov cl, 7
    mov ah, 1
    int 10h
    mov ax, [blinkerTime]
    push ax
    call Delay
    dec dx
    cmp dx, 0
    jne BlinkerLoop
    ;Change text mode to 0 in order to disable intensity/blinking:
    mov ax, 1003h
    mov bx, 0 ;disable blinking
    int 10h
    ;Show standard blinking text cursor:
    mov ch, 6
    mov cl, 7
    mov ah, 1
    int 10h
    ;hide cursor
```

```
    mov cx, 2607h
    mov ah, 1
    int 10h
    ret
endp EndGame

proc PrintScoreAndOptions
    ;Procedure prints score and game options
    push ax
    push bx
    push dx

    call ClearScreen

    ;Move cursor to a specific location using BIOS Interrupt 10h
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 4h ;row
    mov dl, 15h ;column
    int 10h
    ;Print Score OutLine
    mov dx, offset dispScore
    mov ah, 09
    int 21h
```

```
    ;Print Options
    mov dx, offset op1
    mov ah, 09
    int 21h

    ;Move cursor to the place after the word 'score: ' and print the score
    call PrintScore
    ;hide cursor
    mov cx, 2607h
    mov ah, 1
    int 10h

    ;Check if down key or up key is pressed and take action accordingly
WaitForAKey:
    mov ah, 1
    int 16h
    jz WaitForAKey
    mov ah, 0
    int 16h
    cmp ah, 1h ;if esc pressed quit
    je ExitAfterGame
    cmp ah, 1Ch ;if enter is pressed take action
    je Action
```

```
    cmp ah, 50h ;if down key is pressed
    jne CheckIfUp
    ;change option
    mov [opState], 2
    jmp UpdateEndScreen
CheckIfUp:
    cmp ah, 48h ;if up key is pressed
    jne WaitForAKey ;if none of these buttons were clicked wait for a good click (either enter, up or down)
    ;change option
    mov [opState], 1
UpdateEndScreen:
    call ClearScreen ;clear the present screen and type it again updated
    ;Move cursor to a specific location using BIOS Interrupt 10h
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 4h ;row
    mov dl, 15h ;column
    int 10h
    ;Print Score OutLine
    mov dx, offset dispScore
    mov ah, 09
    int 21h

    ;Change option if nessesary
```

```
    cmp [opState], 2
   je op2State
op1State:
    mov dx, offset op1
    jmp prin
op2State:
    mov dx, offset op2
prin:
    mov ah, 09
    int 21h

    ;Move cursor to the place after the word 'score: ' and print the score
    call PrintScore
    ;hide cursor
    mov cx, 2607h
    mov ah, 1
    int 10h
    jmp WaitForAKey

Action:
    cmp [opState], 2
    je ExitAfterGame
    jmp Replay
ExitAfterGame:
```

```
    mov ax, 4c00h
    int 21h
Replay:
    ;call ClearScreen
    pop dx
    pop bx
    pop ax
    ret
endp PrintScoreAndOptions


proc PrintScore
    ;Procedure breaks the score into chars and prints them one after the other
    ;The assumption is that the score isn't bigger than 9999h, therefore not bigger than the score: 39321d

    push ax
    push bx
    push cx
    push dx

    xor dx, dx ;clean the register so it can save the digits
    ;The Score is printed from end to start
    mov cx, 5 ;divide number 5 times by 10
    mov ax, [score] ;ax now stores the game score
```

```
    mov [newScore], ax ;newScore is initialized with score so it can be changed while dividing without harming the program
Divideby10AndPrint:
    mov ax, [newScore]
    mov bx, 10d
    div bx ;ax = dx:ax div bx, dx = dx:ax mod bx
    mov [newScore], ax
    push dx ;save the remainder (the digit) in dx
    ;Move cursor to the last digit in [score]
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 6d ;row of the word 'Score' :
    mov dl, [cursorPosition] ;columns after the word 'Score' :
    int 10h
    pop dx
    ;print digit that came from the divider
    add dx, '0' ;dx holds the digit of the number
    mov ah, 2
    int 21h
    xor dx, dx
    dec [cursorPosition] ;change cursorPosition place to last digit in [score]
    loop Divideby10AndPrint

    mov [cursorPosition], 44d ;keep the cursorPosition in the right starting position
```

```
    pop dx
    pop cx
    pop bx
    pop ax
    ret
endp PrintScore

proc ResetGame
    ;Procedure resets all game variables to default

    ;Clear all registers
    xor ax, ax
    xor bx, bx
    xor cx, cx
    xor dx, dx
    mov [mode], 1
    mov [optState], 1
    mov [lose], 0
    mov [randomLetter], 0
    mov [randomColumn], 0
    mov [randomColor], 0
    mov [rowNumber], 0
    mov [columnNumber], 0
    mov [delayTime], 18 ;Default delay time is about a second
```

```
    mov [pressedKey], 0
    mov [row], 0 ;initialize the first line
    mov [position], 0
    mov [score], 0
    mov [newScore], 0
    mov [points], 5 ;Default points is for Easy mode
    ;initialize the letters and columns arrays
    mov cx, 7
ResetLoop:
    mov [position], cl
    dec [position]
    mov bx, offset lettersArray
    add bl, [position]
    mov [byte ptr bx], 0 ;set all letters in the array to zero
    mov bx, offset columnArray
    add bl, [position]
    mov [byte ptr bx], 0 ;set all columns in the column array to zero
    loop ResetLoop
    mov [position], 0

    ret
endp ResetGame

start:
```

```
    mov ax, @data
    mov ds, ax

    call PrintMainMenu

StartNewGame:
    call PrintGameOptionsMenu
    call ClearScreen

    mov bx, offset RandomMemoryPlace
    ;hide cursor
    mov cx, 2607h
    mov ah, 1
    int 10h
    ;generate first random number and random line and print it in a random color for game to begin
    call GetRandomLetter
    call GetrandomColumn
    call GetRandomColor
    ;Move cursor to a specific location using BIOS Interrupt 10h
    mov ah, 2h
    mov bh, 0h ;page 0
    mov dh, 0h ;first row
    mov dl, [randomColumn] ;random column
    int 10h
```

```
;Print letter in a new line each time by scrolling up
mov ah, 7h
mov al, 4h ;number of lines to scroll
mov bh, [randomColor] ;color attribute of text and background
mov ch, 0 ;y coordinate of top left
mov cl, 0 ;x coordinate of top left
mov dh, 24d ;y coordinate of lower right
mov dl, 79d ;x coordinate of lower right
int 10h
;Print Random Letter:
mov al, [randomLetter]
mov dl, al
mov ah, 2
int 21h
;save first letter and its status
mov bx, offset lettersArray
mov al, [randomLetter]
mov [bx], al ;al now stores the first randomLetter generated
mov bx, offset columnArray
mov al, [randomColumn]
mov [bx], al ;al now stores the first matching column to the first randomLetter generated
mov ax, [delayTime]
push ax
call Delay
```

```
    mov bx, offset RandomMemoryPlace
Game:
    call GetRandomLetter
    call GetrandomColumn
    call GetRandomColor
    xor ax, ax
    mov al, [randomColor]
    push ax
    mov al, [randomLetter]
    push ax
    mov al, [randomColumn]
    push ax
    call PrintLetter
    cmp [lose], 1
    je Loser
    ;Check if a key was clicked
    mov ah, 1
    int 16h
    jz ContinuePlaying
    mov ah, 0
    int 16h
    cmp ah, 1h ; if esc clicked Quit
    je exit
    mov ah, 0 ;ah stores the scan code of the key therefore it should be cleaned
```

```
    push ax ;al stores the pressed key ASCII number
    call CheckPressedKey
    cmp [lose], 1
    je Loser
    jmp ContinuePlaying
Loser:
    call EndGame
    call PrintScoreAndOptions
    jmp NewGame
ContinuePlaying:
    ;inc bx
    mov ax, [delayTime]
    push ax
    call Delay
    jmp Game ;repeat cycle until end of game
NewGame:
    call ResetGame
    jmp StartNewGame
exit:
    mov ax, 4c00h
    int 21h
END start
```