Lesson 2

```
7
8   SECTION .data
9   msg     db      'Hello World!', 0Ah
10
11  SECTION .text
12  global  _start
13
14  _start:
15
16      mov     edx, 13
17      mov     ecx, msg
18      mov     ebx, 1
19      mov     eax, 4
20      int     80h
21
22      mov     ebx, 0      ; return 0 status on exit - 'No Errors'
23      mov     eax, 1      ; invoke SYS_EXIT (kernel opcode 1)
24      int     80h
```

Output:

Hello World!

Lesson 3

```
7
8   SECTION .data
9   msg     db      'Hello, brave new world!', 0Ah ; we can modify this now without having to update anywhere else in the prog
10
11  SECTION .text
12  global  _start
13
14  _start:
15
16      mov     ebx, msg        ; move the address of our message string into EBX
17      mov     eax, ebx        ; move the address in EBX into EAX as well (Both now point to the same segment in memory)
18
19  nextchar:
20      cmp     byte [eax], 0   ; compare the byte pointed to by EAX at this address against zero (Zero is an end of string de
21      jz      finished        ; jump (if the zero flagged has been set) to the point in the code labeled 'finished'
22      inc     eax             ; increment the address in EAX by one byte (if the zero flagged has NOT been set)
23      jmp     nextchar        ; jump to the point in the code labeled 'nextchar'
24
25  finished:
26      sub     eax, ebx        ; subtract the address in EBX from the address in EAX
27                              ; remember both registers started pointing to the same address (see line 15)
28                              ; but EAX has been incremented one byte for each character in the message string
29                              ; when you subtract one memory address from another of the same type
30                              ; the result is number of segments between them - in this case the number of bytes
31
32      mov     edx, eax        ; EAX now equals the number of bytes in our string
33      mov     ecx, msg        ; the rest of the code should be familiar now
34      mov     ebx, 1
35      mov     eax, 4
36      int     80h
37
38      mov     ebx, 0
39      mov     eax, 1
40      int     80h
```

Output:

Hello, brave new world!

Lesson 4

```
7
8   SECTION .data
9   msg     db      'Hello, brave new world!', 0Ah
10
11  SECTION .text
12  global  _start
13
14  _start:
15
16      mov     eax, msg        ; move the address of our message string into EAX
17      call    strlen          ; call our function to calculate the length of the string
18
19      mov     edx, eax        ; our function leaves the result in EAX
20      mov     ecx, msg        ; this is all the same as before
21      mov     ebx, 1
22      mov     eax, 4
23      int     80h
24
25      mov     ebx, 0
26      mov     eax, 1
27      int     80h
28
29  strlen:                     ; this is our first function declaration
30      push    ebx             ; push the value in EBX onto the stack to preserve it while we use EBX in this function
31      mov     ebx, eax        ; move the address in EAX into EBX (Both point to the same segment in memory)
32
33  nextchar:                   ; this is the same as lesson3
34      cmp     byte [eax], 0
35      jz      finished
36      inc     eax
37      jmp     nextchar
38
39  finished:
40      sub     eax, ebx
41      pop     ebx             ; pop the value on the stack back into EBX
42      ret                     ; return to where the function was called
```

Output:

HelloWorld.asm:30: error: instruction not supported in 64-bit mode
HelloWorld.asm:41: error: instruction not supported in 64-bit mode

Lesson 5

```asm
 8   int slen(String message)
 9  ; String length calculation function
10  slen:
11      push    ebx
12      mov     ebx, eax
13
14  nextchar:
15      cmp     byte [eax], 0
16      jz      finished
17      inc     eax
18      jmp     nextchar
19
20  finished:
21      sub     eax, ebx
22      pop     ebx
23      ret
24
25
26  ;--------------------------------
27  ; void sprint(String message)
28  ; String printing function
29  sprint:
30      push    edx
31      push    ecx
32      push    ebx
33      push    eax
34      call    slen
35
36      mov     edx, eax
37      pop     eax
38
39      mov     ecx, eax
40      mov     ebx, 1
41      mov     eax, 4
42      int     80h
43
44      pop     ebx
45      pop     ecx
```

```asm
 8  %include        'functions.asm'                     ; include our external file
 9
10  SECTION .data
11  msg1    db      'Hello, brave new world!', 0Ah       ; our first message string
12  msg2    db      'This is how we recycle in NASM.', 0Ah  ; our second message string
13
14  SECTION .text
15  global _start
16
17  _start:
18
19      mov     eax, msg1    ; move the address of our first message string into EAX
20      call    sprint       ; call our string printing function
21
22      mov     eax, msg2    ; move the address of our second message string into EAX
23      call    sprint       ; call our string printing function
24
25      call    quit         ; call our quit function
```

Lesson 6

```asm
 7
 8  %include        'functions.asm'
 9
10  SECTION .data
11  msg1    db      'Hello, brave new world!', 0Ah, 0h      ; NOTE the null terminating byte
12  msg2    db      'This is how we recycle in NASM.', 0Ah, 0h  ; NOTE the null terminating byte
13
14  SECTION .text
15  global _start
16
17  _start:
18
19      mov     eax, msg1
20      call    sprint
21
22      mov     eax, msg2
23      call    sprint
24
25      call    quit
```

Lesson 7:

```
6
7
8    ;-------------------------------------------
9    ; int slen(String message)
10   ; String length calculation function
11 ▾ slen:
12       push    ebx
13       mov     ebx, eax
14
15 ▾ nextchar:
16       cmp     byte [eax], 0
17       jz      finished
18       inc     eax
19       jmp     nextchar
20
21 ▾ finished:
22       sub     eax, ebx
23       pop     ebx
24       ret
25
26
27   ;-------------------------------------------
28   ; void sprint(String message)
29   ; String printing function
30 ▾ sprint:
31       push    edx
32       push    ecx
33       push    ebx
34       push    eax
35       call    slen
36
37       mov     edx, eax
38       pop     eax
39
40       mov     ecx, eax
41       mov     ebx, 1
42       mov     eax, 4
43       int     80h
44
45       pop     ebx
46
```

```
6
7
8    %include     'functions.asm'
9
10   SECTION .data
11   msg1    db      'Hello, brave new world!', 0h       ; NOTE we have removed the line feed character 0Ah
12   msg2    db      'This is how we recycle in NASM.', 0h  ; NOTE we have removed the line feed character 0Ah
13
14   SECTION .text
15   global  _start
16
17   _start:
18
19       mov     eax, msg1
20       call    sprintLF    ; NOTE we are calling our new print with linefeed function
21
22       mov     eax, msg2
23       call    sprintLF    ; NOTE we are calling our new print with linefeed function
24
25       call    quit
```

Output:

HelloWorld.asm

Lesson 8

```
7
8    %include     'functions.asm'
9
10   SECTION .text
11   global  _start
12
13   _start:
14
15       pop     ecx             ; first value on the stack is the number of arguments
16
17 ▾ nextArg:
18       cmp     ecx, 0h         ; check to see if we have any arguments left
19       jz      noMoreArgs      ; if zero flag is set jump to noMoreArgs label (jumping over the end of the loop)
20       pop     eax             ; pop the next argument off the stack
21       call    sprintLF        ; call our print with linefeed function
22       dec     ecx             ; decrease ecx (number of arguments left) by 1
23       jmp     nextArg         ; jump to nextArg label
24
25 ▾ noMoreArgs:
26       call    quit
```

Output:

HelloWorld.asm:8:

Lesson 9

```asm
7
8   %include        'functions.asm'
9
10  SECTION .data
11  msg1        db      'Please enter your name: ', 0h      ; message string asking user for input
12  msg2        db      'Hello, ', 0h                       ; message string to use after user has entered their name
13
14  SECTION .bss
15  sinput:     resb    255                                 ; reserve a 255 byte space in memory for the users input string
16
17  SECTION .text
18  global  _start
19
20  _start:
21
22      mov     eax, msg1
23      call    sprint
24
25      mov     edx, 255        ; number of bytes to read
26      mov     ecx, sinput     ; reserved space to store our input (known as a buffer)
27      mov     ebx, 0          ; read from the STDIN file
28      mov     eax, 3          ; invoke SYS_READ (kernel opcode 3)
29      int     80h
30
31      mov     eax, msg2
32      call    sprint
33
34      mov     eax, sinput     ; move our buffer into eax (Note: input contains a linefeed)
35      call    sprint          ; call our print function
36
37      call    quit
```

Output:

HelloWorld.asm:

Lesson 10

```asm
8   %include        'functions.asm'
9
10  SECTION .text
11  global  _start
12
13  _start:
14
15      mov     ecx, 0          ; ecx is initalised to zero.
16
17  nextNumber:
18      inc     ecx             ; increment ecx
19
20      mov     eax, ecx        ; move the address of our integer into eax
21      add     eax, 48         ; add 48 to our number to convert from integer to ascii for printing
22      push    eax             ; push eax to the stack
23      mov     eax, esp        ; get the address of the character on the stack
24      call    sprintLF        ; call our print function
25
26      pop     eax             ; clean up the stack so we don't have unneeded bytes taking up space
27      cmp     ecx, 10         ; have we reached 10 yet? compare our counter with decimal 10
28      jne     nextNumber      ; jump if not equal and keep counting
29
30      call    quit
```

Output:

HelloWorld.asm