15.2.1

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    nElements,10    @ number of elements in array
        .equ    intArray,-52    @ array beginning
        .equ    locals,40       @ space for local vars

@ Constant program data
        .section .rodata
        .align  2
prompt:
        .asciz          "Enter an integer:\n"
display:
        .asciz          "In reverse order:\n"

@ The program
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 16      @ space for saving regs
        str     r4, [sp, 8]     @ save r4
        str     r5, [sp, 4]     @     r5
        str     fp, [sp, 8]     @     fp
        str     lr, [sp, 12]    @     lr
```

```
        str     lr, [sp, 12]    @     lr
        add     fp, sp, 12      @ set our frame pointer
        sub     sp, sp, locals  @ for the array

        add     r4, fp, intArray  @ address of array beginning
        mov     r5, 0           @ index = 0;
fillLoop:
        cmp     r5, nElements   @ all filled?
        bge     allFull         @ yes
        ldr     r0, promptAddr  @ no, prompt user
        bl      writeStr
        bl      getDecInt       @ get integer
        lsl     r1, r5, 2       @ offset is 4 * index
        str     r0, [r4, r1]    @ at index-th element
        add     r5, r5, 1       @ index++;
        b       fillLoop
allFull:

        ldr     r0, displayAddr @ nice message
        bl      writeStr

        add     r4, fp, intArray  @ address of array beginning
        mov     r5, 9           @ index = 9;
printLoop:
        lsl     r1, r5, 2       @ no, offset is 4 * index
        ldr     r0, [r4, r1]    @ at index-th element
        bl      putDecInt       @ print integer
        bl      newLine
        subs    r5, r5, 1       @ index--;
        bge     printLoop
```

```
        bge     printLoop
allDone:
        mov     r0, 0           @ return 0;
        add     sp, sp, locals  @ deallocate local var
        ldr     r4, [sp, 8]     @ restore r4
        ldr     r5, [sp, 4]     @     r5
        ldr     fp, [sp, 8]     @        fp
        ldr     lr, [sp, 12]    @        lr
        add     sp, sp, 16      @ restore sp
        bx      lr              @ return

promptAddr:
        .word   prompt
displayAddr:
        .word   display
```

15.5.1

structPass.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "theTagStruct.s"  @ theTag struct defs.
        .equ    y,-36           @ y struct
        .equ    x,-24           @ x struct
        .equ    locals,24       @ space for the structs

@ Constant program data
        .section .rodata
        .align  2
displayX:
        .asciz          "x fields:\n"
displayY:
        .asciz          "y fields:\n"

@ The Program
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer
        sub     sp, sp, locals  @ for the structs
```

```
        sub     sp, sp, locals  @ for the structs

@ fill the x struct
        add     r0, fp, x       @ address of x struct
        bl      getStruct

@ fill the y struct
        add     r0, fp, y       @ address of y struct
        bl      getStruct

@ display x struct
        ldr     r0, displayXaddr
        bl      writeStr
        add     r0, fp, x       @ address of x struct
        bl      putStruct
        bl      newLine

@ display y struct
        ldr     r0, displayYaddr
        bl      writeStr
        add     r0, fp, y       @ address of y struct
        bl      putStruct
        bl      newLine
```

```
        bl      putStruct
        bl      newLine

        mov     r0, 0           @ return 0;
        add     sp, sp, locals  @ deallocate local var
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @          fp
        ldr     lr, [sp, 12]    @          lr
        add     sp, sp, 16      @          sp
        bx      lr              @ return

        .align  2
@ addresses of messages
displayXaddr:
        .word   displayX
displayYaddr:
        .word   displayY
```

getStruct.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "theTagStruct.s"  @ theTag struct defs.

@ Constant program data
        .section .rodata
        .align  2
charPrompt:
        .asciz          "Enter a character: "
intPrompt:
        .asciz          "Enter an integer: "

@ The program
        .text
        .align  2
        .global getStruct
        .type   getStruct, %function
getStruct:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer

        mov     r4, r0          @ pointer to the struct
```

```
        ldr     r0, charPromptAddr
        bl      writeStr        @ ask for a char
        bl      getChar         @ get it
        strb    r0, [r4, aChar] @ aStruct->aChar = firstChar;

        ldr     r0, intPromptAddr
        bl      writeStr        @ ask for a char
        bl      getDecInt       @ get it
        str     r0, [r4, anInt] @ aStruct->anInt = aNumber;

        ldr     r0, charPromptAddr
        bl      writeStr        @ ask for a char
        bl      getChar         @ get it
        strb    r0, [r4, anotherChar] @ aStruct->anotherChar = secondChar

        mov     r0, 0           @ return 0;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return

        .align  2
@ addresses of messages
charPromptAddr:
        .word   charPrompt
intPromptAddr:
        .word   intPrompt
```

putStruct.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "theTagStruct.s"  @ theTag struct defs.

@ Constant program data
        .section .rodata
        .align  2
dispAChar:
        .asciz          "       aChar = "
dispAnInt:
        .asciz          "       anInt = "
dispOtherChar:
        .asciz          "   anotherChar = "

@ The Program
        .text
        .align  2
        .global putStruct
        .type   putStruct, %function
putStruct:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer
```

```
        mov     r4, r0          @ pointer to the struct

        ldr     r0, dispACharAddr  @ display aChar
        bl      writeStr
        ldrb    r0, [r4, aChar]
        bl      putChar
        bl      newLine
        ldr     r0, dispAnIntAddr  @ display anInt
        bl      writeStr
        ldr     r0, [r4, anInt]
        bl      putDecInt
        bl      newLine
        ldr     r0, dispOtherCharAddr @ display anotherChar
        bl      writeStr
        ldrb    r0, [r4, anotherChar]
        bl      putChar
        bl      newLine

        mov     r0, 0           @ return 0;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return

        .align  2
@ addresses of messages
dispACharAddr:
        .word   dispAChar
dispAnIntAddr:
        .word   dispAnInt
dispOtherCharAddr:
        .word   dispOtherChar
```

getChar.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    maxChars,2      @ max input chars
        .equ    inputChar,-12   @ for input chars
        .equ    locals,8        @ space for local vars

@ The program
        .text
        .align  2
        .global getChar
        .type   getChar, %function
getChar:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @    and lr
        add     fp, sp, 4       @ set our frame pointer
        sub     sp, sp, locals  @ for the string

        add     r0, fp, inputChar  @ place to store input
        mov     r1, maxChars    @ limit input length
        bl      readLn

        ldrb    r0, [fp, inputChar] @ return inputChar

        add     sp, sp, locals  @ deallocate local var
        ldr     fp, [sp, 0]     @ restore caller fp
        ldr     lr, [sp, 4]     @       lr
        add     sp, sp, 8       @   and sp
        bx      lr              @ return
```

15.7.1

incFraction.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    x,-12           @ x fraction object
        .equ    locals,8        @ space for fraction

@ The program
        .text
        .align  2
        .global main
        .type   main, %function
main:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @   and lr
        add     fp, sp, 4       @ set our frame pointer
        sub     sp, sp, locals  @ for the structs

@ construct the fraction
        add     r0, fp, x       @ address of x struct
        bl      fractionConstr

@ get user input
        add     r0, fp, x       @ get user values
        bl      fractionGet

@ add 1
        add     r0, fp, x       @ add 1 to it
        mov     r1, 1
        bl      fractionAddInt

@ display result
        add     r0, fp, x       @ get user values
        bl      fractionDisplay

        mov     r0, 0           @ return 0;
        add     sp, sp, locals  @ deallocate local var
        ldr     fp, [sp, 0]     @ restore caller fp
        ldr     lr, [sp, 4]     @       lr
        add     sp, sp, 8       @   and sp
        bx      lr              @ return
```

fractionObject.s

```
@ fraction object definition
        .equ    num,0    @ numerator
        .equ    den,4    @ denominator
```

fractionConstr.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "fractionObject.s"  @ fraction object defs.

@ The code
        .text
        .align  2
        .global fractionConstr
        .type   fractionConstr, %function
fractionConstr:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @   and lr
        add     fp, sp, 4       @ set our frame pointer

        mov     r1, 1           @ reasonable numerator
        str     r1, [r0, num]

        mov     r1, 2           @ reasonable denominator
        str     r1, [r0, den]

        mov     r0, 0           @ return 0;
        ldr     fp, [sp, 0]     @ restore caller fp
        ldr     lr, [sp, 4]     @       lr
        add     sp, sp, 8       @   and sp
        bx      lr              @ return
```

fractionGet.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "fractionObject.s"  @ fraction object defs.

@ Constant program data
        .section .rodata
        .align  2
numPrompt:
        .asciz      "  Enter numerator: "
denPrompt:
        .asciz      "Enter denominator: "

@ The code
        .text
        .align  2
        .global fractionGet
        .type   fractionGet, %function
fractionGet:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer

        mov     r4, r0          @ pointer to the object
```

fractionGet.s

```
        ldr     r0, numPromptAddr
        bl      writeStr        @ ask for numerator
        bl      getDecInt       @ get it
        str     r0, [r4, num]   @ store at this->num

        ldr     r0, denPromptAddr
        bl      writeStr        @ ask for denominator
        bl      getDecInt       @ get it
        str     r0, [r4, den]   @ store at this->den

        mov     r0, 0           @ return 0;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return

        .align  2
@ addresses of messages
numPromptAddr:
        .word   numPrompt
denPromptAddr:
        .word   denPrompt
```

fractionAddInt.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "fractionObject.s"  @ fraction object defs.

@ The code
        .text
        .align  2
        .global fractionAddInt
        .type   fractionAddInt, %function
fractionAddInt:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer

        mov     r4, r0          @ this pointer

        ldr     r0, [r4, den]   @ get denominator
        mul     r2, r1, r0      @ integer X denominator
        ldr     r0, [r4, num]   @ get numerator
        add     r2, r2, r0      @ numerator + (int X den)
        str     r2, [r4, num]   @ save new numerator

        mov     r0, 0           @ return 0;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return
```

fractionDisplay.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .include "fractionObject.s"  @ fraction object defs.

@ The code
        .text
        .align  2
        .global fractionDisplay
        .type   fractionDisplay, %function
fractionDisplay:
        sub     sp, sp, 16      @ space for saving regs
                                @ (keeping 8-byte sp align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @      fp
        str     lr, [sp, 12]    @      lr
        add     fp, sp, 12      @ set our frame pointer

        mov     r4, r0          @ this pointer

        ldr     r0, [r4, num]   @ display numerator
        bl      putDecInt
        mov     r0, '/          @ slash for fraction
        bl      putChar
        ldr     r0, [r4, den]   @ display denominator
        bl      putDecInt
        bl      newLine

        mov     r0, 0           @ return 0;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @         fp
        ldr     lr, [sp, 12]    @         lr
        add     sp, sp, 16      @         sp
        bx      lr              @ return
```

putChar.s

```
        .cpu    cortex-a53
        .fpu    neon-fp-armv8
        .syntax unified         @ modern syntax

@ Constants for assembler
        .equ    STDOUT,1
        .equ    theChar,-5      @ for  string
        .equ    locals,8        @ space for local var

@ The code
        .text
        .align  2
        .global putChar
        .type   putChar, %function
putChar:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @   and lr
        add     fp, sp, 4       @ set our frame pointer
        sub     sp, sp, locals

        strb    r0, [fp, theChar]  @ write needs address
        mov     r0, STDOUT      @ write to screen
        add     r1, fp, theChar    @ address of theChar
        mov     r2, 1           @ write 1 byte
        bl      write

        mov     r0, 0           @ return 0;
        add     sp, sp, locals  @ deallocate local var
        ldr     fp, [sp, 0]     @ restore caller fp
        ldr     lr, [sp, 4]     @       lr
        add     sp, sp, 8       @   and sp
        bx      lr              @ return
```