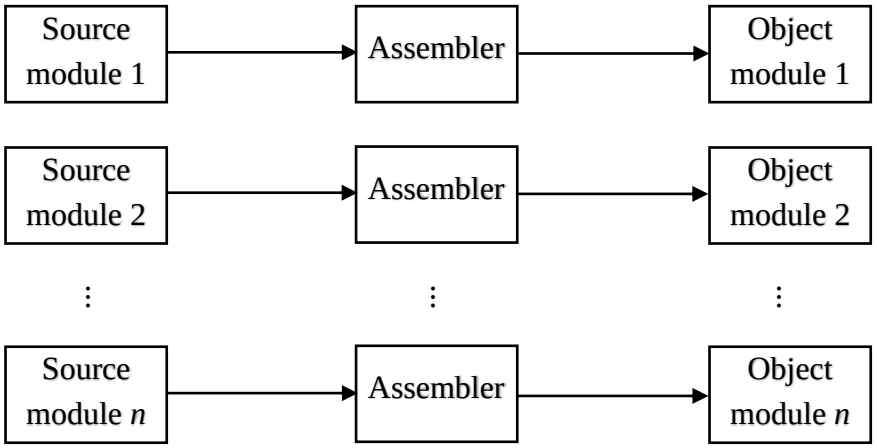
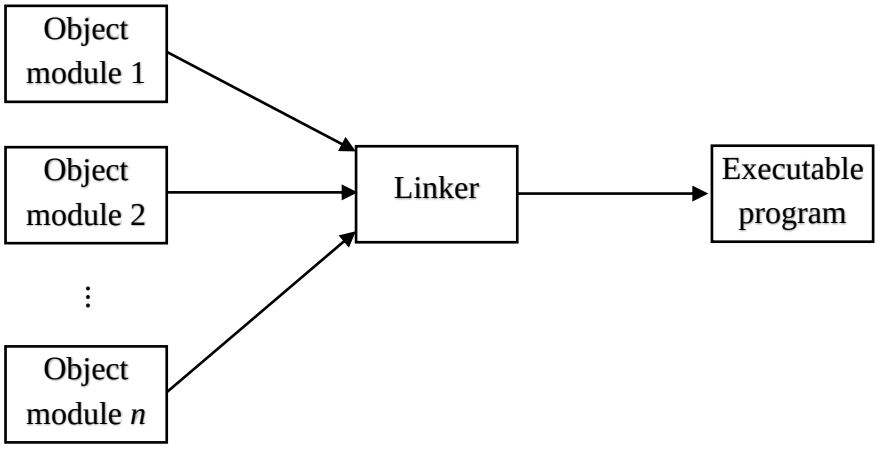
12 Linking

Separate Assembly

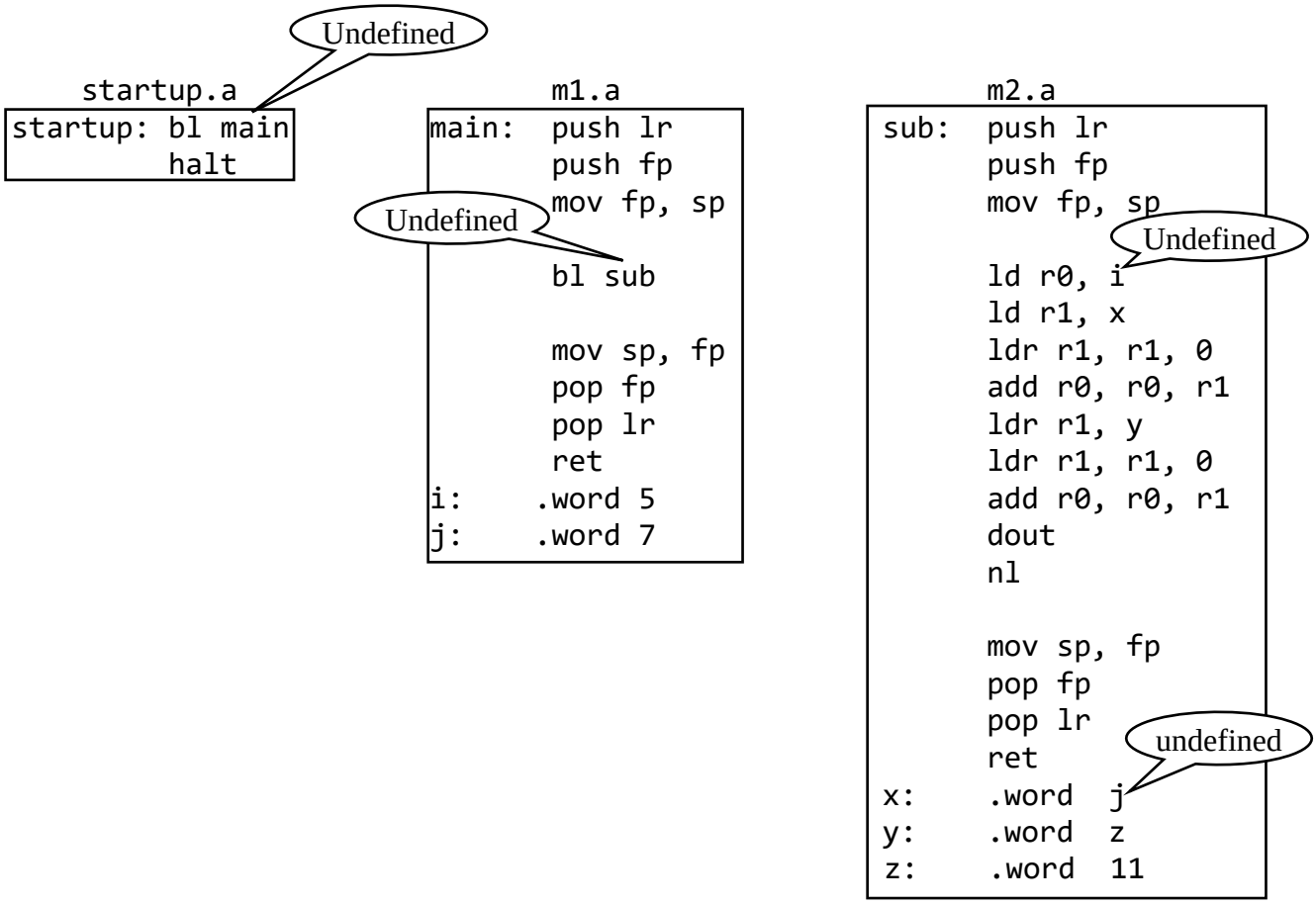




We use the "`.a`", "`.o`", and "`.e`" extensions for the assembly language source module files, the object module files, and the executable program file, respectively.

Requirements for Linking

Buggy modules:



Corrected modules

startup.a

```
        .start startup
        .extern main
startup: bl main
        halt
```

m1.a

```
        .extern sub
        .global i
        .global j
        .global main
main:   push lr
        push fp
        mov fp, sp

        bl sub

        mov sp, fp
        pop fp
        pop lr
        ret
i:      .word 5
j:      .word 7
```

m2.a

```
      .extern i
      .extern j
      .global sub
sub:  push lr
      push fp
       mov fp, sp

      ld r0, i
      ld r1, x
      ldr r1, r1, 0
      add r0, r0, r1
      ld r1, y
      ldr r1, r1, 0
      add r0, r0, r1
      dout
      nl

      mov sp, fp
      pop fp
      pop lr
      ret
x:    .word  j
y:    .word  z
z:    .word  11
```

```
lcc startup.a
lcc m1.a
lcc m2.a


lcc startup.o m1.o m2.o


lcc link.e
```

Overview of the Linking Process

`bl` instruction in `startup.a` produced by the assembler is

```
0100 1 00000000000
```

Address must be adjusted by linker

```
y          .word z                  ; assembled to 18 decimal
z          .word 11
```

Address must be adjusted by linker

```
0000000000010010                    ; assembled to addr rel to begin of module
```

The linker must adjust this address so that it is the address of `z` *relative to the beginning of the linked program*. In the *linked program*, the address of `z` is

the address of the beginning of the `m2` module in the linked program (000c hex)

\+

the address of `z` relative to the beginning of the `m2` module (0012 hex)

Linking Process in Detail

Header entries:

startup.o	m1.o	m2.o
S 0000	G 0000 main	G 0000 sub
E 0000 main	E 0003 sub	e 0003 i
	G 0008 i	V 0010 j
	G 0009 j	A 0011

- An S entry provides the entry point for the program.
- A G entry provides the address of a global label.
- An E entry provides the external label referenced and the address of the 11-bit external reference.
- An e entry provides the external label referenced and the address of the 9-bit external reference.
- A V entry provides the external label referenced and the address of the 16-bit external reference.
- An A entry provides the address of a 16-bit local reference.
- A C entry separates the header from the machine code.

S table

```
0000
```

G table

```
0002 main
000a i
000b j
000c sub
```

E table

```
0000 main
0005 sub
```

e table

```
000f i
```

V table

```
001c j
```

A table

```
001d 000c
```

Address of module that contains the A entry

Machine code array

```
Loc  | array | 
0000 | 4800 | startup:  bl main
0001 | f000 |           halt
0002 | ae01 | main:     push lr
0003 | aa01 |           push fp
0004 | 1ba0 |           mov fp, sp
0005 | 4800 |           bl sub
0006 | 1d60 |           mov sp,fp
0007 | aa02 |           pop fp
0008 | ae02 |           pop lr
0009 | c1c0 |           ret
000a | 0005 | i:        .word 5
000b | 0007 | j:        .word 7
000c | ae01 | sub:      push lr
000d | aa01 |           push fp
000e | 1ba0 |           mov fp, sp
000f | 2000 |           ld r0, i
0010 | 220b |           ld r1, x
0011 | 6240 |           ldr r1, r1, 0
0012 | 1001 |           add r0, r0, r1
0013 | 2209 |           ld r1, y
0014 | 6240 |           ldr r1, r1, 0
0015 | 1001 |           add r0, r0, r1
0016 | f002 |           dout
0017 | f001 |           nl
0018 | 1d60 |           mov sp, fp
0019 | aa02 |           pop fp
001a | ae02 |           pop lr
001b | c1c0 |           ret
001c | 0000 | x:        .word  j
001d | 0012 | y:        .word  z
001e | 000b | z:        .word  11
```

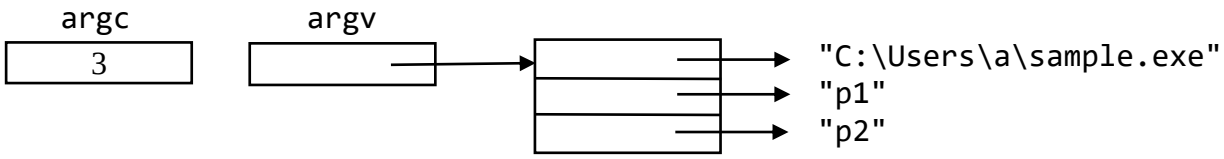
Linked Module

```
Header:
    o
    S  0000
    G  0002 main
    G  000a i
    G  000b j
    G  000c sub
    A  001c
    A  001d
    C

Code:
    0: 4801 f000 ae01 aa01 1ba0 4806 1d60 aa02
    8: ae02 c1c0 0005 0007 ae01 aa01 1ba0 21fa
   10: 220b 6240 1001 2209 6240 1001 f002 f001
   18: 1d60 aa02 ae02 c1c0 000b 001e 000b
```

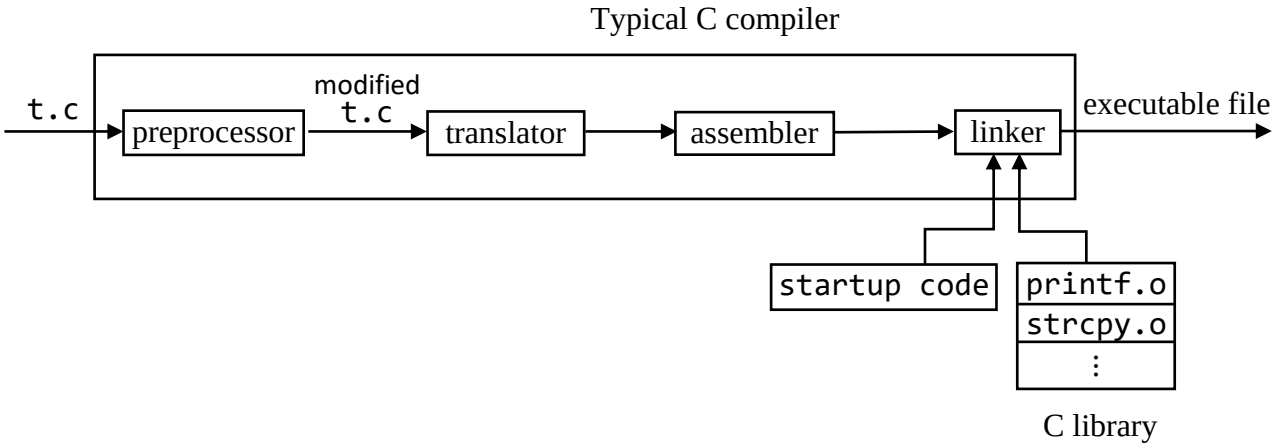
Startup Code

```
sample p1 p2
```




```
int main(int argc, char **argv);
```

```
int main(int argc, char *argv[]);
```




```
lcc t.a
lcc su.o t.o -o t.e
lcc t.e p1 p2
```

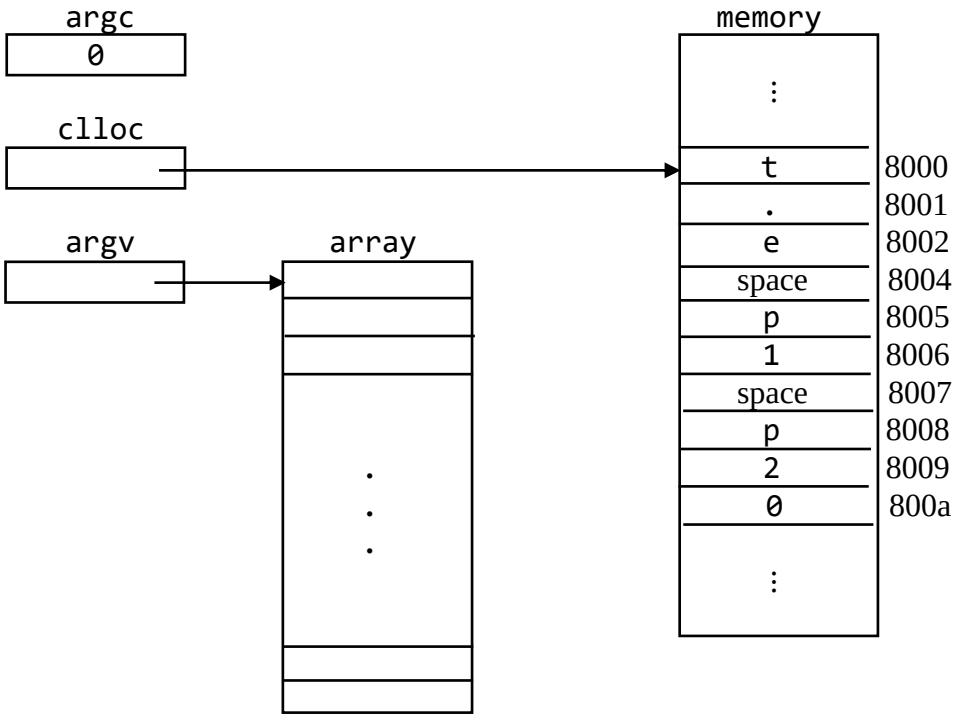
```
 1 ; su.a  start-up code that configures argc and argv
 2           .start su        ; makes su the entry point
 3           .extern main     ; needed to link to main
 4 sig:      .word 'Z'        ; signature for this module
 5
 6 su:       ld r3, clloc     ; r3 points to command line
 7           lea r4, array    ; r4 point to argv array
 8           mov r5, 0        ; r5 is arg counter
 9
10 getarg:   str r3, r4, 0    ; store arg addr in array
11           add r4, r4, 1    ; inc pointer to array
12           add r5, r5, 1    ; increment arg counter
13
14 nextchar: add r3, r3, 1    ; move com line pointer
15           ldr r0, r3, 0    ; get char from com line
16           cmp r0, 0        ; is it the null char
17           brz cldone       ; branch if end of command line
18           ld r1, blank     ; load blank
19           cmp r0, r1       ; compare char and blank
20           brnz nextchar    ; branch if not blank
21
22           mov r0, 0        ; get null char
23           str r0, r3, 0    ; overlay blank with null char
24           add r3, r3, 1    ; advance ptr to next arg
25           br getarg
26
27 cldone:   st r5, argc      ; store arg count in argc
28           mov sp, 0        ; initialize sp and fp
29           mov fp, 0
30
31           ld r0, argv      ; call main passing it argc, argv
32           push r0
33           ld r0, argc
34           push r0
35           bl  main         ; program must have main function
36           add sp, sp, 2    ; remove parms, r0 has return code
37
38           ld r1, sig       ; get signature
39           ld r2, sigcopy   ; get original signature
40           cmp r1, r2       ; compare signatures
41           brz alldone      ; branch if the same
42           lea r0, m1       ; get address of error message
43           sout
45 alldone:  halt
46 argc:     .word 0
47 argv:     .word array
48 array:    .zero 20        ; argv array
49 m1:       .string "\nStart-up code corrupted\n"
50 clloc:    .word 0x8000    ; command line location
51 blank:    .word ' '
52 sigcopy:  .word 'Z'
```

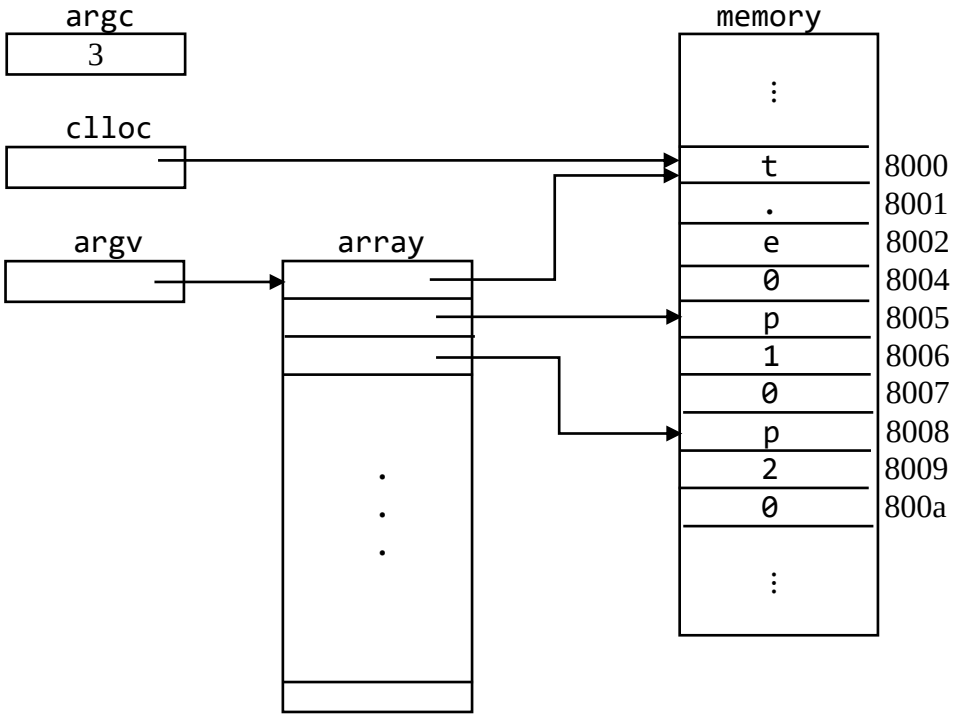
Can use brne here

Can use bre here

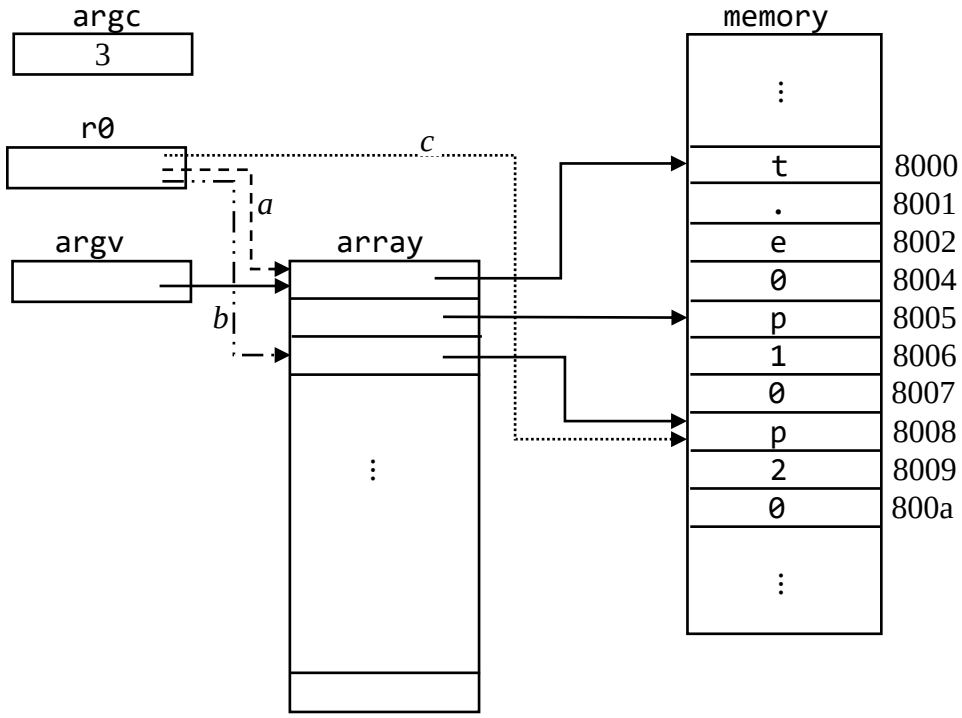
```
lcc t.e p1 p2
```

Then on entry into the startup code, these are the structures that exist:




argc
3
clloc
argv
array
memory
t
.
e
0
p
1
0
p
2
0
8000
8001
8002
8004
8005
8006
8007
8008
8009
800a

```
 1 ; ex1201.a  Command line arguments
 2                           ; #include <stdio.h>
 3           .global main    ; int main(int argc, char *argv[])
 4 main:     push lr         ; {
 5           push fp
 6           mov fp, sp
 7
 8           ldr r0, fp, 2   ;    int i = argc-1;
 9           add r0, r0, -1
10           push r0
11
12 @L0:      ldr r0, fp, -1  ;    while (i >= 0)
13           cmp r0, 0       ;    {
14           brn @L1
15
16           ldr r0, fp, 3   ;       printf("%s\n", argv[i]);
17           ldr r1, fp, -1
18           add r0, r0, r1
19           ldr r0, r0, 0
20           sout
21           nl
22
23           ldr r0, fp, -1  ;       i--;
24           sub r0, r0, 1
25           str r0, fp, -1
26           br @L0
27
28 @L1:                       ;    }
29
30           mov r0, 0       ;    return 0;
31           mov sp, fp
32           pop fp
33           pop lr
34           ret
35                          ; }
```


argc
3
r0
argv
array
a
b
c
memory
t
.
e
0
p
1
0
p
2
0
8000
8001
8002
8004
8005
8006
8007
8008
8009
800a

Separately-Compiled C Modules

z1.c

```
1 // z1.c
2 void f(void);
3 int x = 5;
4 int main()
5 {
6    f();
7    return 0;
8 }
```

z2.c

```
 1 // z2.c
 2 #include <stdio.h>
 3 extern int x;
 4 int y = 7;
 5 void g(void)
 6 {
 7    printf("%d\n", y);
 8 }
 9 //===================
10 void f(void)
11 {
12    printf("%d\n", x);
13    g();
14 }
```

```
gcc z1.c z2.c
```

z1.a

```
1  ; z1.a
2                    ; void f(void);
3       .global x    ; int x = 5;
4  x:   .word 5
5
6       .global main ; int main()
7  main:push lr      ; {
8       push fp      ;
9       mov fp, sp
10
11      bl f         ;    f();
12
13      mov r0, 0    ;    return 0;
14      mov sp, fp
15      pop fp
16      pop lr
17      ret
18
19      .extern f    ; }
20
```

z2.a

```
; z2.a
                ; #include <stdio.h>
   .extern x    ; extern int x;
   .global y    ; int y = 7
y:  .word 7

   .global g    ; void g(void)
g:  push lr      ; {
   push fp
   mov fp, sp

   ld r0, y     ;    printf("%d\n", y);
   dout
   nl

   mov sp, fp ; }
   pop fp
   pop lr
   ret
;===================================
   .global f    ; void f(void)
f: push lr      ; {
   push fp
   mov fp, sp

   ld r0, x     ;    printf("%d\n", x);
   dout
   nl

   bl g         ;    g();

   mov sp, fp ;}
   pop fp
   pop lr
   ret
```

```
 1 // z2.c modifed
 2 #include <stdio.h>
 3 int x;
 4 static int y = 7;        // keyword static gives y file scope
 5 static void g(void)      // keyword static give g file scope
 6 {
 7    printf("%d\n", y);
 8 }
 9//===================
10 void f(void)
11 {
12    printf("%d\n", x);
13    g();
14 }
```

The *only* effect of the `static` keyword on lines 4 and 5 is to suppress the compiler from generating the `.global` directives for `y` and `g`, thereby given `y` and `g` file scope.