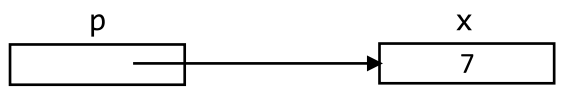
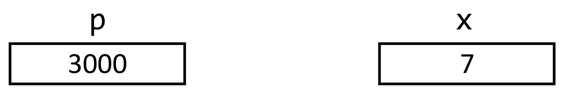
7 Pointers

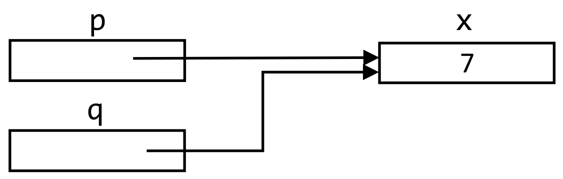
Pointers in C

```
int p;      // p has type int

int *p;     // p has type int pointer
```





```
p = &x;     // &x is the address of x


q = p;
```



```
y = *p;     // assigns y the value that p points to


*p = 10;    // assigns 10 to the location that p points to


y = x;      // directly accessing x
y = *p;     // accessing x via a pointer


p = p + 1; // cannot do this in Java
```

Pointers to Global Variables

```
x:        .word 0
p:        .word 0


p = &x;
```

is

```
          lea r0, x        ; get address of x
          st r0, p         ; store address of x in p

y = *p

          ld r0, p         ; load r0 with address in p
          ldr r0, r0, 0    ; load from address given by r0 +
          st r0, y

*p = 10;
          mov r0, 10
          ld r1, p         ; load r1 with address in p
          str r0, r1, 0    ; store 10 at address given by r1 + 0
```

Example with Pointers to Global Variables

```
; ex0701.a  Pointers to global variables
startup:  bl main
          halt
;==============================================================
                           ; #include <stdio.h>
p:        .word 0          ; int *p, x = 7;
x:        .word 7

main:     push lr          ; int main()
          push fp          ; {
          mov fp, sp

          lea r0, x        ;    p = &x;
          st r0, p

          ld r0, p         ;    printf("%d\n", *p);
          ldr r0, r0, 0
          dout r0
          nl

          mov r0, 8        ;    *p = 8;
          ld r1, p
          str r0, r1, 0

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

          mov r0, 0        ;    return 0;
          mov sp, fp
          pop fp
          pop lr
          ret
                           ; }
```

Pointers to Dynamic Local Variables

```
          ldr r0, fp, -1   // loads r0 from local var at offset -1
```

If, instead, we want to load `r0` with the *address* of the stack item at offset -1, we use

```
          add r0, fp, -1   // loads r0 with addr of local var at offset -1
```

Dereferencing

```
          ldr r0, fp, -1   ; get local pointer into r0
```

We then load `r0` from the address that `r0` points to:

```
          ldr r0, r0, 0    ; load r0 from address given by r0 + 0
```

To store a value in a location that a local variable points to,

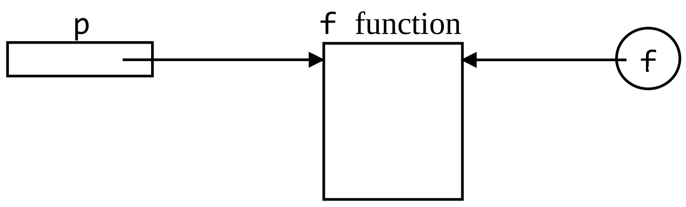
```
         mov r0, 10        ; get the value to be stored
         ldr r1, fp, -1    ; get the pointer into r1
         str r0, r1, 0     ; store value in r0 at address given by r1 + 0
```

Example with Pointers to Local Variables

```
; ex0702.a  Pointers to local variables
startup:  bl main
          halt
;===============================================================
                           ; #include <stdio.h>
                           ; int main()
main:     push lr          ; {
          push fp
          mov fp, sp

          sub sp, sp, 1    ;    int *p, x = 7;
          mov r0, 7                Get address of
          push r0                   local var x

          add r0, fp, -2   ;    p = &x;
          str r0, fp, -1

          ldr r0, fp, -1   ;    printf("%d\n", *p);
          ldr r0, r0, 0
          dout r0                Dereference p
          nl

          mov r0, 8        ;    *p = 8;
          ldr r1, fp, -1
          str r0, r1, 0           Dereference p

          ldr r0, fp, -2   ;    printf("%d\n", x);
          dout r0
          nl

          mov r0, 0        ;    return 0;
          mov sp, fp
          pop fp
          pop lr
          ret
                           ; }
```

Pointers to Functions

```
     void f(int x, int y)

     f(1, 2);   // calls f function


     p = f;     // does not call f
```



```
     f(1, 2);
     p(1, 2);


     p = g;     // p now points to the g function
```

But assigning `g` to `f` is not legal because `f` is a constant pointer:

```
     f = g;     // illegal because f always points to the f function
```

Declaring p:

```
     void (*p)(int, int);
```

Incorrect!!!

```
     void *p(int, int);   // parens high precedence that *
```

`p` would then be a function (because the parentheses indicate `p` is a function) that returns

```
     void *
```

Example of Pointers to Functions

```
; ex0703.a  Pointers to functions
startup:  bl main
          halt
;==============================================================
                           ; #include <stdio.h>
sum:      .word 0          ; int sum;
p:        .word 0          ; int (*p)(int, int);
;==============================================================
f:        push lr          ; int f(int x, int y)
          push fp          ; {
          mov fp, sp

          ldr r0, fp, 2    ;    return x+y;
          ldr r1, fp, 3
          add r0, r0, r1
          mov sp, fp
          pop fp
          pop lr
          ret
                           ; }
;==============================================================
main:     push lr           ; int main()
          push fp           ; {
          mov fp, sp

          mov r0, 2         ;    sum = f(1, 2);
          push r0
          mov r0, 1
          push r0
          bl f
          add sp, sp, 2
          st r0, sum


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

          lea r0, f         ;    p = f;
          st r0, p

          mov r0, 2         ;    sum = p(1, 2);
          push r0
          mov r0, 1
          push r0
          ld r0, p
          blr r0
          add sp, sp, 2
          st r0, sum

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

          mov r0, 0         ;    return 0;
          mov sp, fp
          pop fp
          pop lr
          ret
                            ; }
```