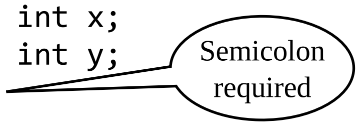
9 Structs

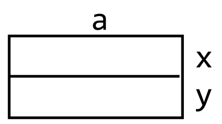
Creating and Accessing Structs

```
struct Point
{
   int x;
   int y;
};
```



```
struct Point a;
```

```
Point a;   // illegal in C, but not in C++.
```



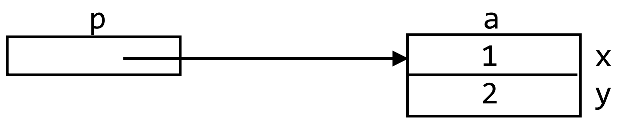
```
a.x = 1;
a.y = 2;
```

```
struct Point     // create new type
{
   int x;
   int y;
} a;             // also declare a
```

```
struct Point *p;
```

```
p = &a;
```

We then get the following configuration:



```
p->y = 3;  // assign 3 to the y field of the struct that p points to
```

```
(*p).y = 3;
```

```
; ex0901.a  Structs
startup:  bl main
          halt
;==============================================================
                           ; #include <stdio.h>
                           ; struct Point
                           ; {
                           ;    int x;
                           ;    int y;
                           ; };

a:        .zero 2          ; struct Point a;
p:        .word 0          ; struct Point *p;

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

          mov r0, 1        ;    a.x = 1;
          st r0, a

          mov r0, 2        ;    a.y = 2;
          st r0, a+1

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

          ld r0, a+1       ;    printf("%d\n", a.y);
          dout r0
          nl

          ld r0, p         ;    printf("%d\n", p->y);
          ldr r0, r0, 1
          dout r0
          nl

          ld r0, p         ;    printf("%d\n", (*p).y);
          ldr r0, r0, 1
          dout r0
          nl

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

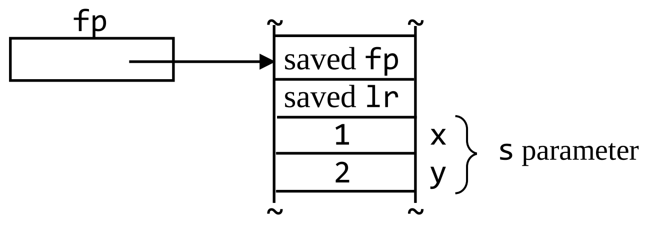
Dynamically Allocating Structs

```c
// ex0902.c  Dynamically allocating structs
#include <stdio.h>
#include <stdlib.h> // required my malloc
struct Point
{
   int x;
   int y;
};
struct Point *p;
//==================================
int main()
{
   p = (struct Point *)malloc(sizeof(struct Point));
   p->y = 5;
   printf("%d\n", p->y);
   return 0;
}
```

```
; ex0902.a  Dynamically allocating structs
startup:  bl main
          halt
;==============================================================
                           ; #include <stdio.h>
                           ; #include <stdlib.h>
                           ; struct Point
                           ; {
                           ;    int x;
                           ;    int y;
                           ; };

p:        .word 0          ; struct Point *p;

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

          mov r1, 2   ; p = (struct Point *)malloc(sizeof(struct Point));
          bl malloc
          st r0, p

          mov r0, 5        ;    p -> y = 5;
          ld r1, p
          str r0, r1, 1

          ld r0, p         ;    printf("%d\n", p -> y);
          ldr r0, r0, 1
          dout r0
          nl

          mov r0, 0        ;    return 0;
          mov sp, fp
          pop fp
          pop lr
          ret
                           ; }
;==============================================================
malloc:   ld r0, @avail    ; get address of next free block
          add r1, r0, r1   ; r1 holds size of allocation
          st r1, @avail    ; update @avail
          ret              ; return address of allocated block
@avail:  .word *+1
```

Passing Structs

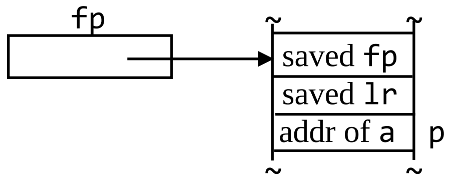
```
// ex0903.c  Passing structs
#include <stdio.h>
struct Point
{
   int x;
   int y;
};
struct Point a;
//================================
void f(struct Point s)
{
   printf("%d %d\n", s.x, s.y);
}
//================================
void g(struct Point *p)
{
   printf("%d %d\n", p->x, p->y);
}
//================================
int main()
{
   a.x = 1;
   a.y = 2;
   f(a);     // pass by value
   g(&a);    // pass by address
}
```



In f, the fields of the s parameter are accessed with a single instruction. For example, the y field is loaded into r0 with

```
        ldr r0, fp, 3
```

The call of g on line 25 passes a by address. The calling sequence for g pushes the address of a onto the stack thereby creating the parameter p in g. Here is a picture of the stack when g is executing:



To access the fields of a via the parameter p in g requires two instructions: one to get the address of a and a second to access the desired field. For example, the y field is loaded into r0 with

```
ldr r1, fp, 2     ; get address of a
ldr r0, r1, 1     ; get y field
```

Rule: Use pass by address to pass a struct or any array.

```
; ex0903.a  Passing structs
startup:  bl main
          halt
;==============================================================
                            ; #include <stdio.h>
                            ; struct Point
                            ; {
                            ;    int x;
                            ;    int y;
                            ; };
a:        .word 0           ; struct Point a;
          .word 0
;==============================================================
f:        push lr           ; void f(struct Point s)
          push fp           ; {
          mov fp, sp

          ldr r0, fp, 2     ;    printf("%d %d\n", s.x, s.y);
          dout r0
          mov r0, ' '
          aout
          ldr r0, fp, 3
          dout r0
          nl

          mov sp, fp        ; }
          pop fp
          pop lr
          ret
;==============================================================
g:        push lr           ; void g(struct Point *p)
          push fp           ; {
          mov fp, sp

          ldr r1, fp, 2     ;    printf("%d %d\n", p->x, p->y);
          ldr r0, r1, 0
          dout r0
          mov r0, ' '
          aout
          ldr r0, r1, 1
          dout r0
          nl

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

Only one instruction needed to access y

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

          mov r0 1          ;    a.x = 1;
          st r0, a

          mov r0, 2         ;    a.y = 2;
          st r0, a+1

          ld r0, a+1        ;    f(a);
          push r0
          ld r0, a
          push r0
          bl f
          add sp, sp, 2

          lea r0, a         ;    g(&a);
          push r0
          bl g
          add sp, sp, 1

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

                            ; }
```