Passing Pointers to Struct

There are a couple of ways of handling structs using pointers:
- (*pointer).attribute
- pointer->attribute

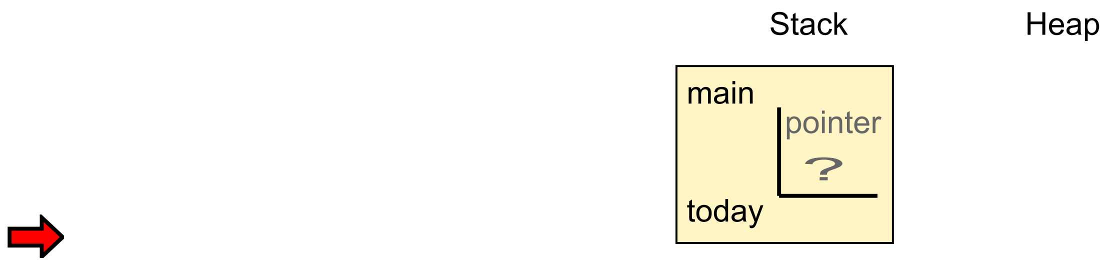
We'll cover the following

- Pointer to a struct

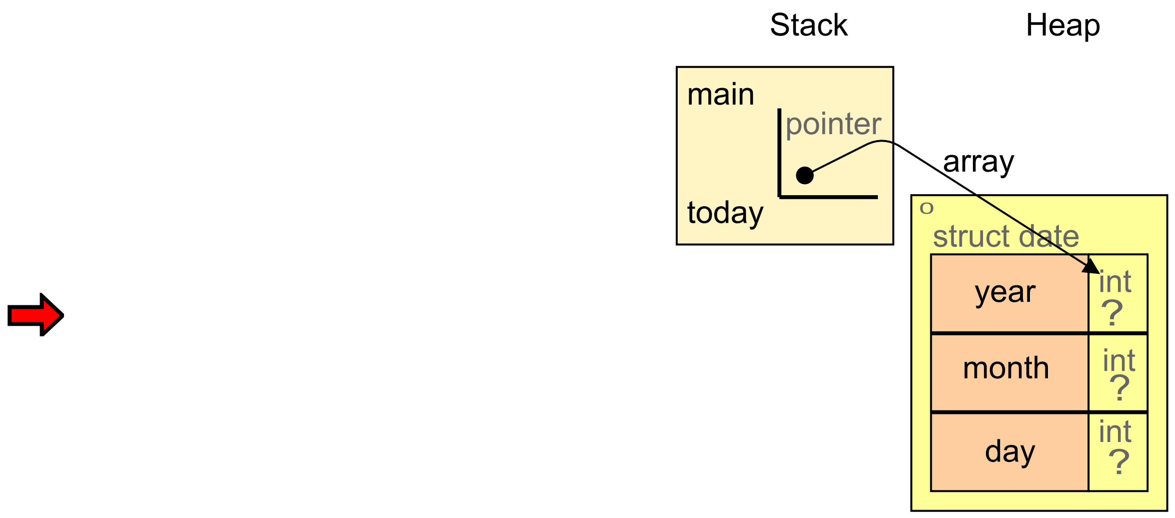
Pointer to a struct

Pointers can also be used to point to a struct. Here is how this would be done:

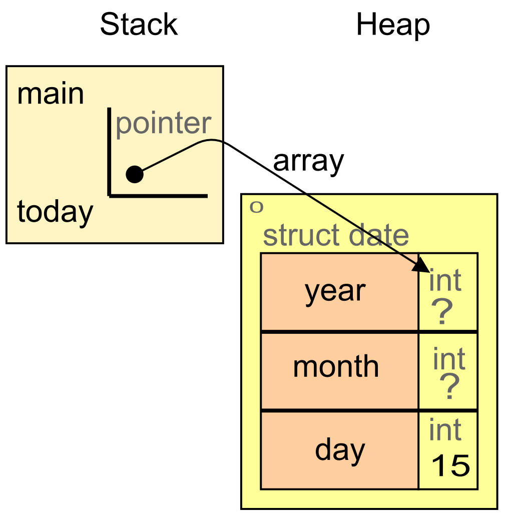
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int year;
  int month;
  int day;
} date;

int main(void) {

  date *today;
  today = (date*)malloc(sizeof(date));

  // the explicit way of accessing fields of our struct
  (*today).day = 15;
  (*today).month = 6;
  (*today).year = 2012;

  // the more readable shorthand way of doing it
  today->day = 15;
  today->month = 6;
  today->year = 2012;

  printf("the date is %d/%d/%d\n", today->day, today->month, today->year);

  free(today);

  return 0;
}
```

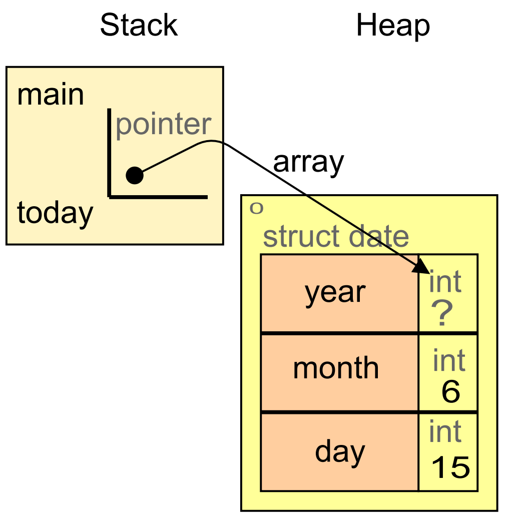


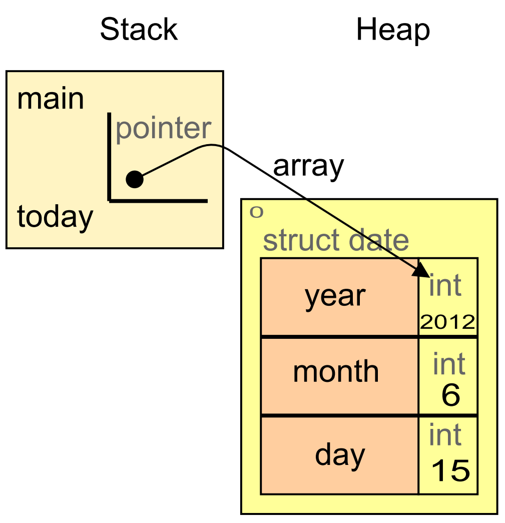
Stack
Heap
main
pointer
?
today

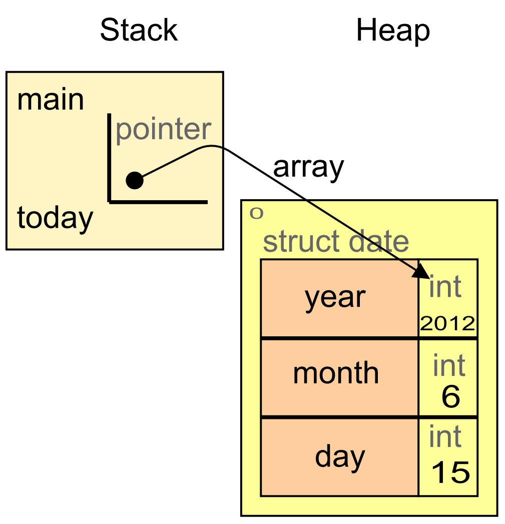


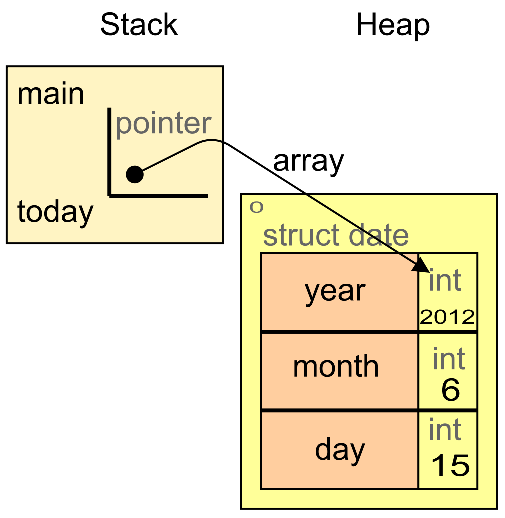
Stack
Heap
main
pointer
today
array
o
struct date
year
int
?
month
int
?
day
int
?

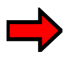

Stack
Heap
main
pointer
today
array
o
struct date
year
int
?
month
int
?
day
int
15

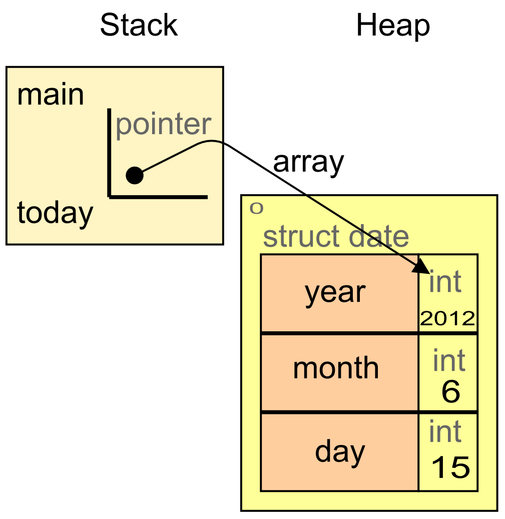

Stack
Heap
main
pointer
today
array
o
struct date
year
int
?
month
int
6
day
int
15

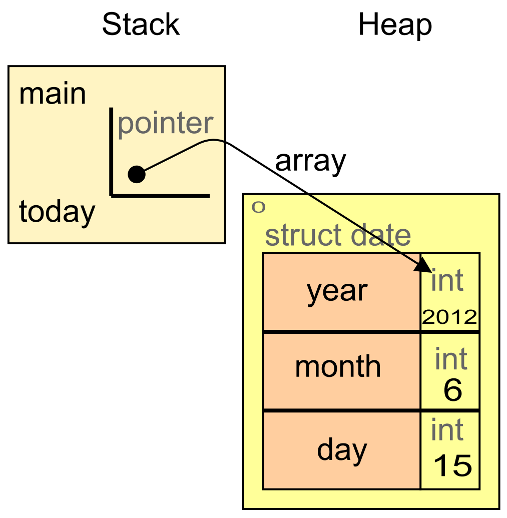

Stack
Heap
main
pointer
today
array
o
struct date
year
int
2012
month
int
6
day
int
15


Stack
Heap
main
pointer
today
array
o
struct date
year
int
2012
month
int
6
day
int
15

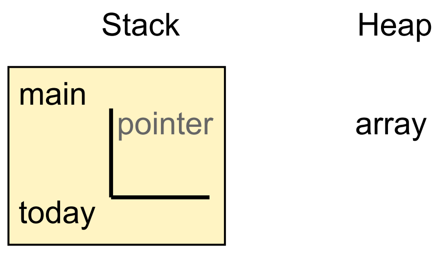

Stack
Heap
main
pointer
today
array
o
struct date
year
int
2012
month
int
6
day
int
15

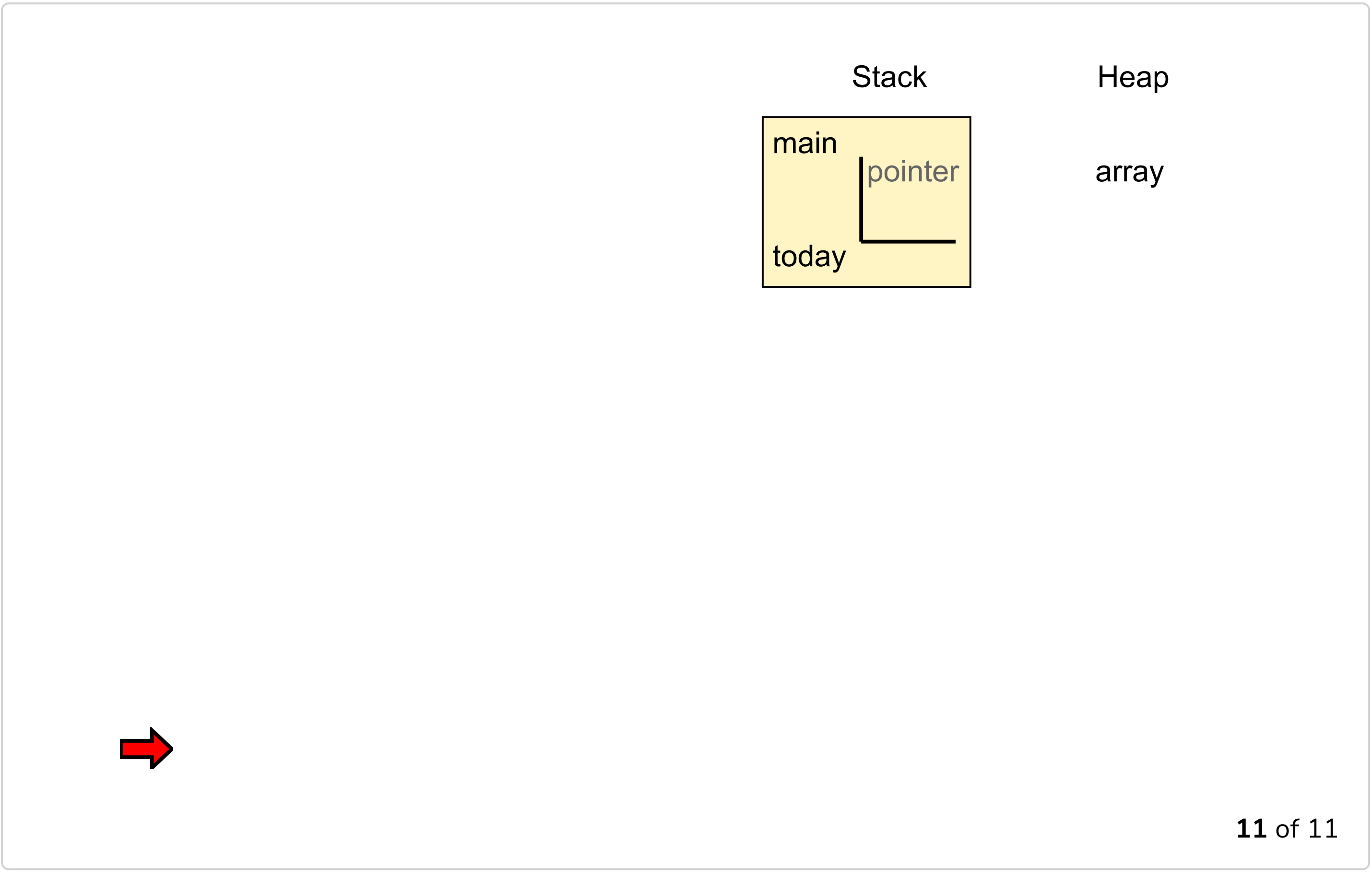



Stack
Heap
main
pointer
today
array
o
struct date
year
int
2012
month
int
6
day
int
15



Output: The date is 15/6/2012





Refer to the slides above while we go through this example step by step. On lines **4-8** we define a struct that contains three `int` values: `year`, `month` and `day`. We use typedef to name our new struct type `date`.

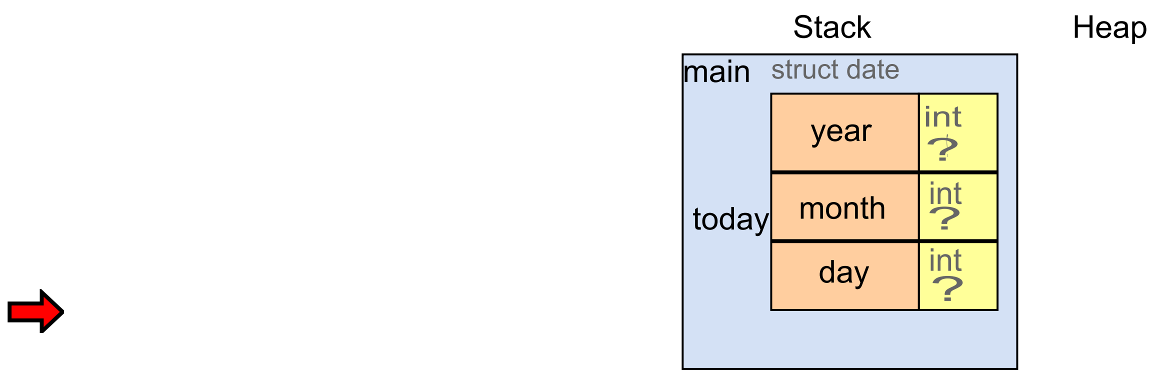
On line **12** we declare a new variable `today` to be a pointer to `date`. On line **13** we use `malloc()` to allocate a block of memory (on the heap) to store one `date` struct.

On lines **16-18** I show how to access fields of our `date` struct, using explicit pointer syntax. So for example the expression `(*today).day` means, dereference the `today` pointer and then access the `day` field of the thing you find there (which will be a `date` struct).

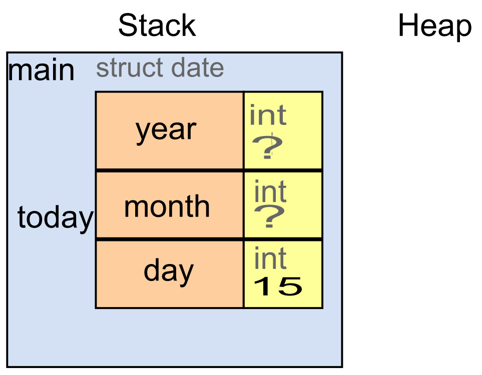
On lines **21-23** I show you the more common (and more readable) shorthand for using pointers with structs.

Just as a reminder: here is how one would do this on the stack instead of the heap:

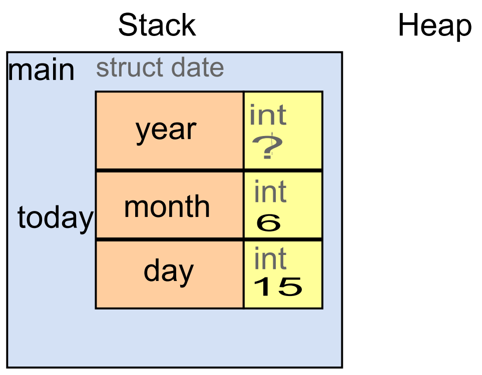
```
#include <stdio.h>

typedef struct {
  int year;
  int year;
  int month;
  int day;
} date;

int main(void) {

  date today;

  today.day = 15;
  today.month = 6;
  today.year = 2012;

  printf("the date is %d/%d/%d\n", today.day, today.month, today.year);

  return 0;
}
```



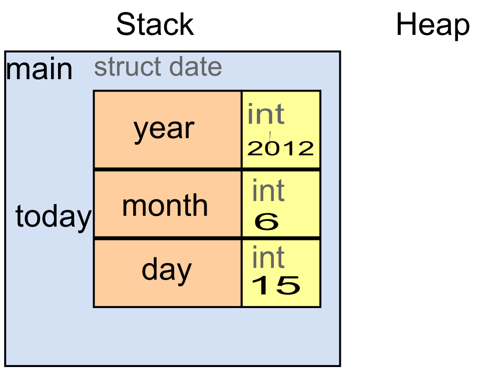
1 of 6

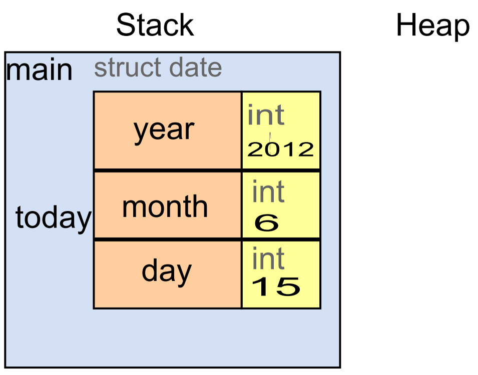


Stack
Heap
main
struct date
today
year
int
?
month
int
?
day
int
15

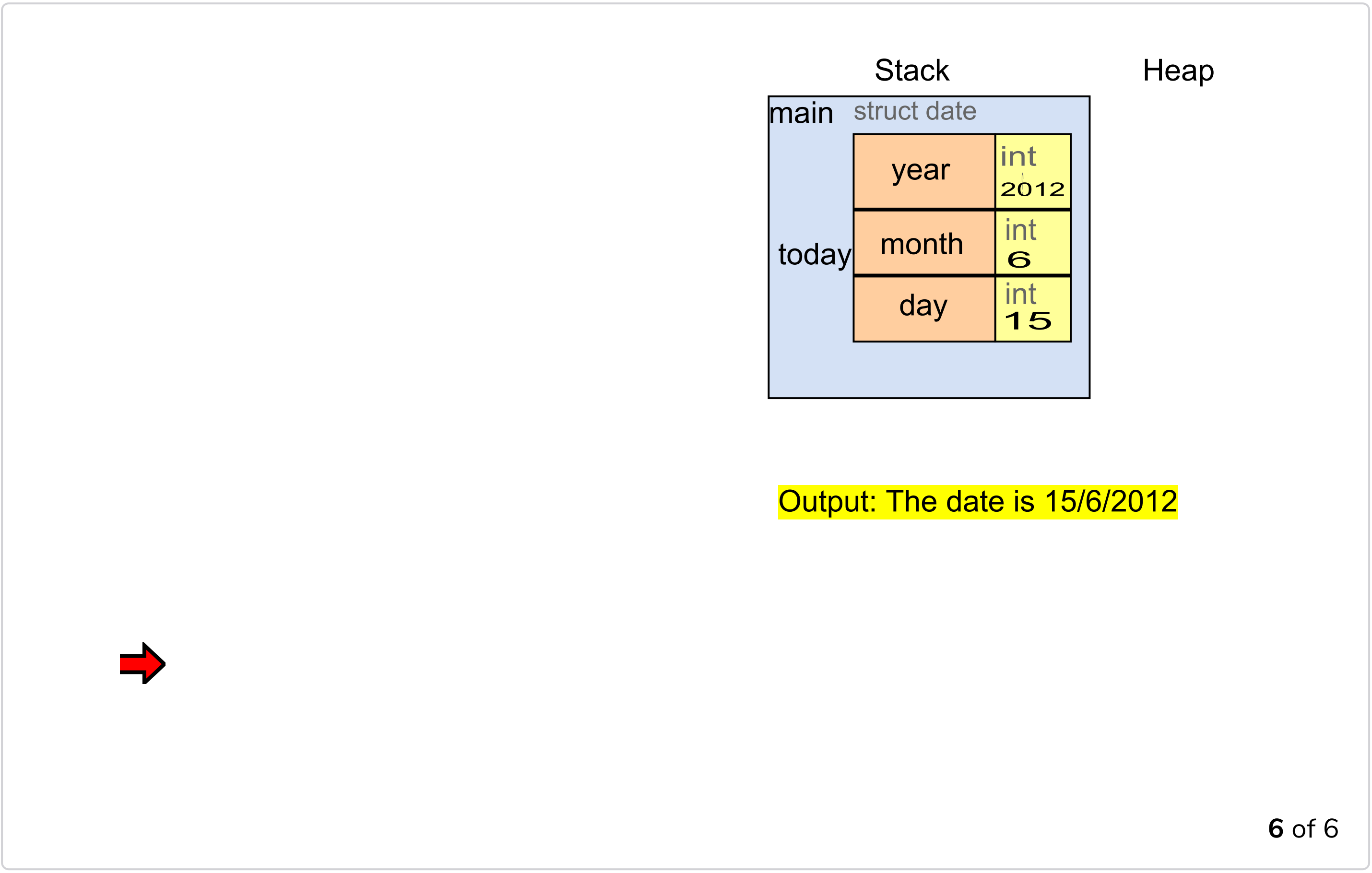


Stack
Heap
main
struct date
today
year
int
?
month
int
6
day
int
15





```
Output: The date is 15/6/2012
```



```
Output: The date is 15/6/2012
```

Gone is all of the pointer stuff, at least on the surface. Under the hood, C is still using pointers to accomplish this.

Now it’s time to move on to how pointers work with functions. Things will get slightly more complex now.