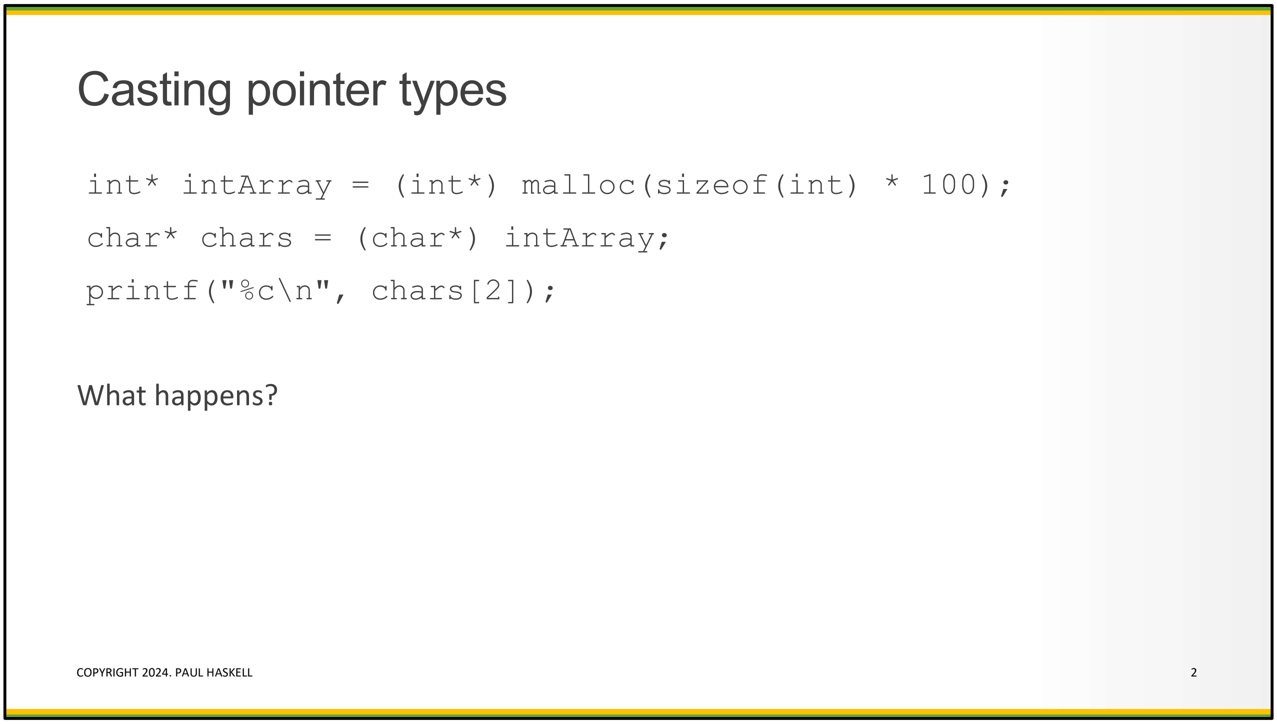
CS221
C and Systems Programming

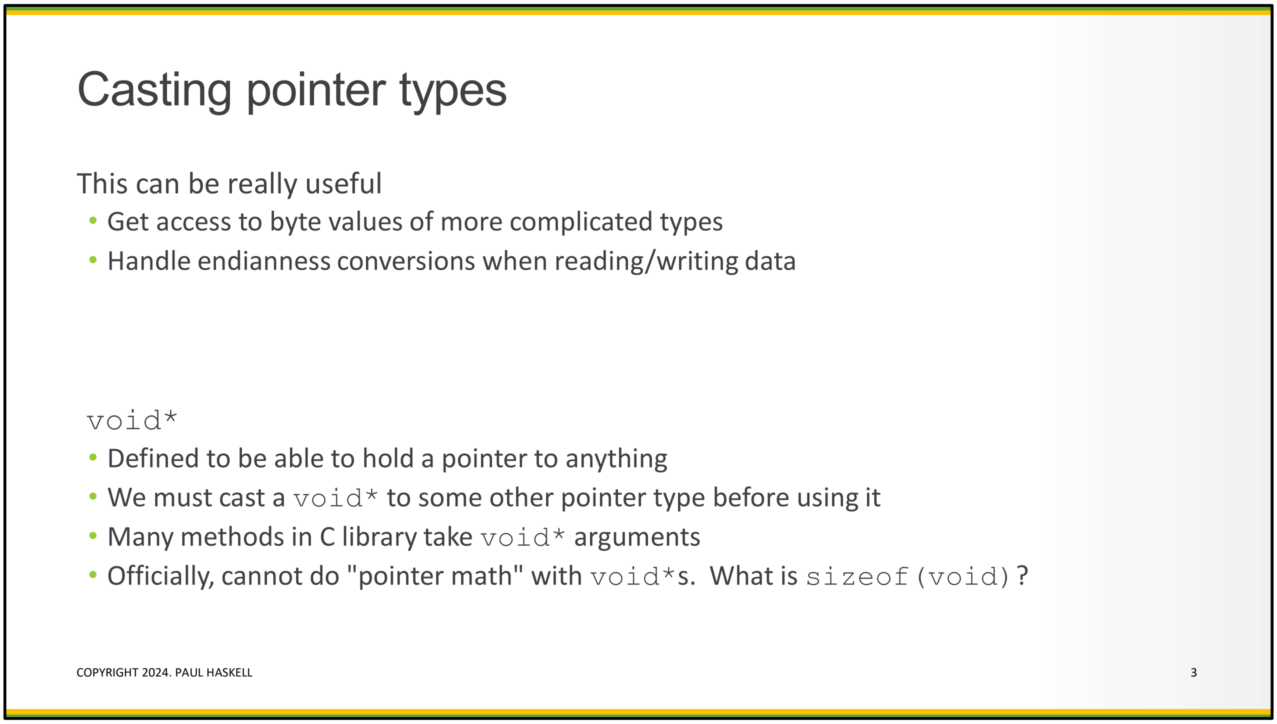


Casting pointer types

```
int* intArray = (int*) malloc(sizeof(int) * 100);
char* chars = (char*) intArray;
printf("%c\n", chars[2]);
```

What happens?

No conversions happen, no code runs. Computer INTERPRETS values in memory as an int, or a char. But values in memory don't change.
See simple.c



Casting pointer types

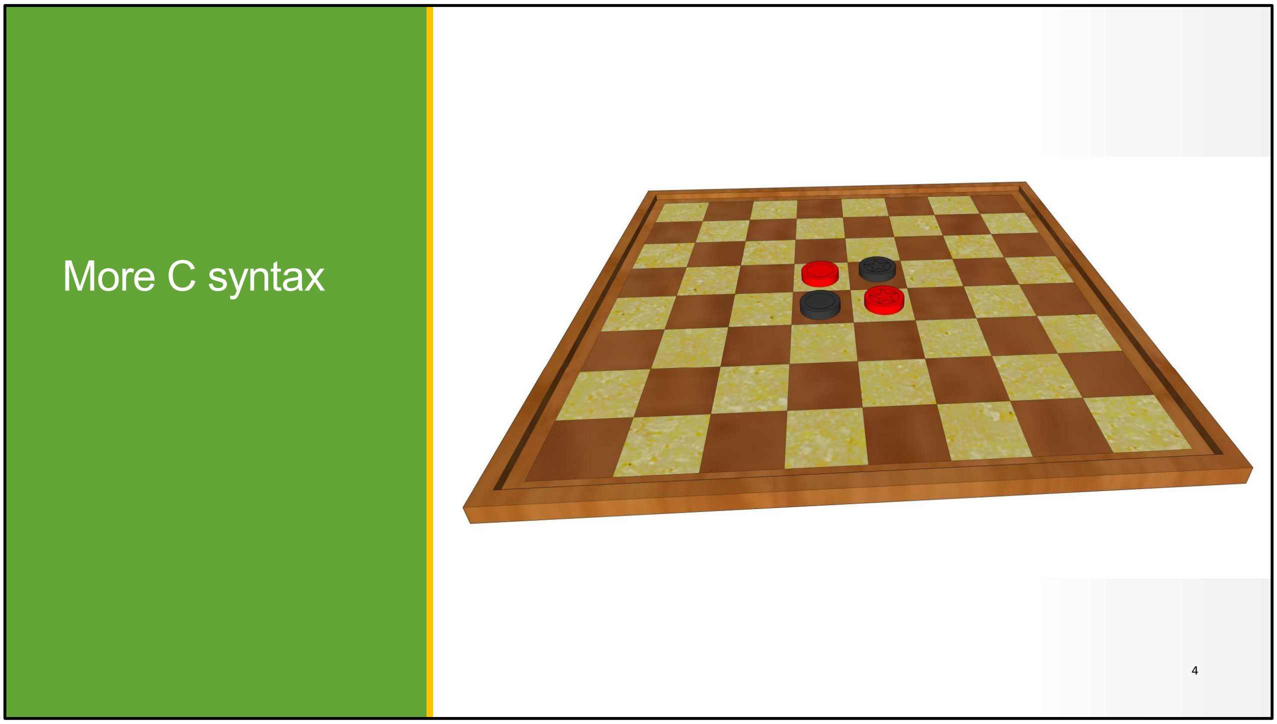
This can be really useful

- Get access to byte values of more complicated types
- Handle endianness conversions when reading/writing data

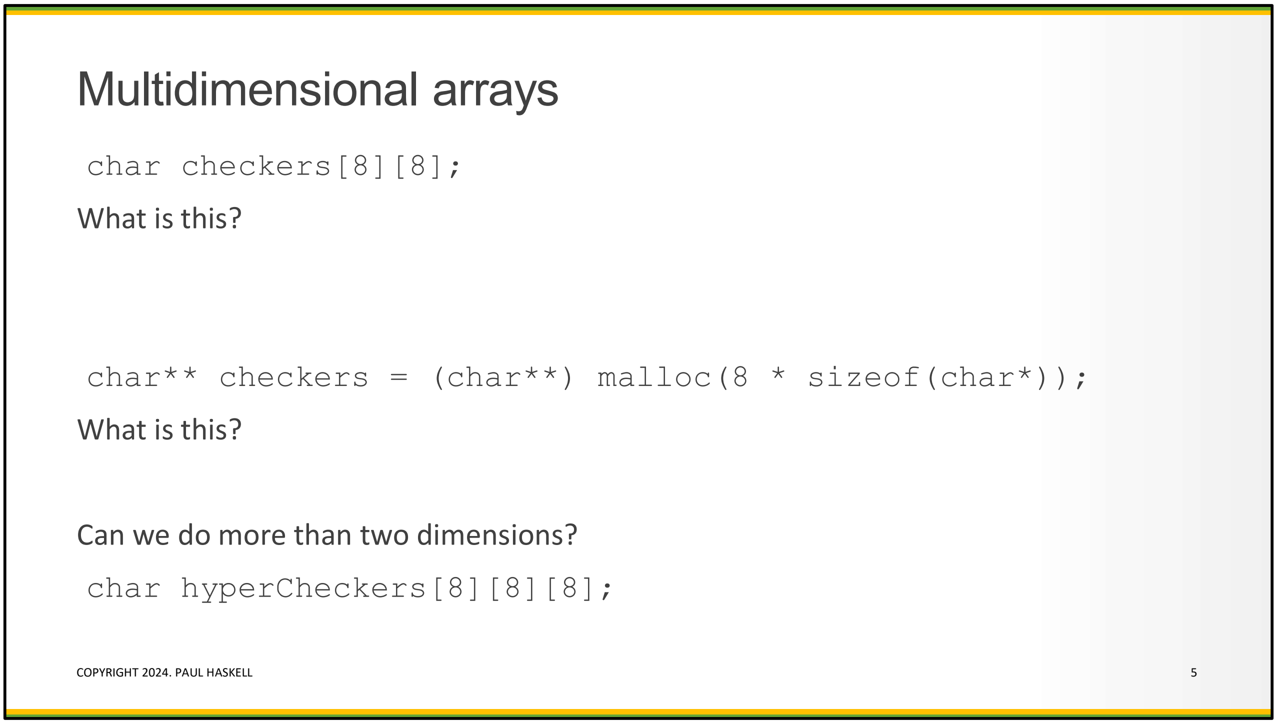
`void*`

- Defined to be able to hold a pointer to anything
- We must cast a `void*` to some other pointer type before using it
- Many methods in C library take `void*` arguments
- Officially, cannot do "pointer math" with `void*`s. What is `sizeof(void)`?

But of course this can be really dangerous!
(GNU defines sizeof(void) == 1, but NOT STANDARD)



More C syntax



Multidimensional arrays

```
char checkers[8][8];
```

What is this?

```
char** checkers = (char**) malloc(8 * sizeof(char*));
```

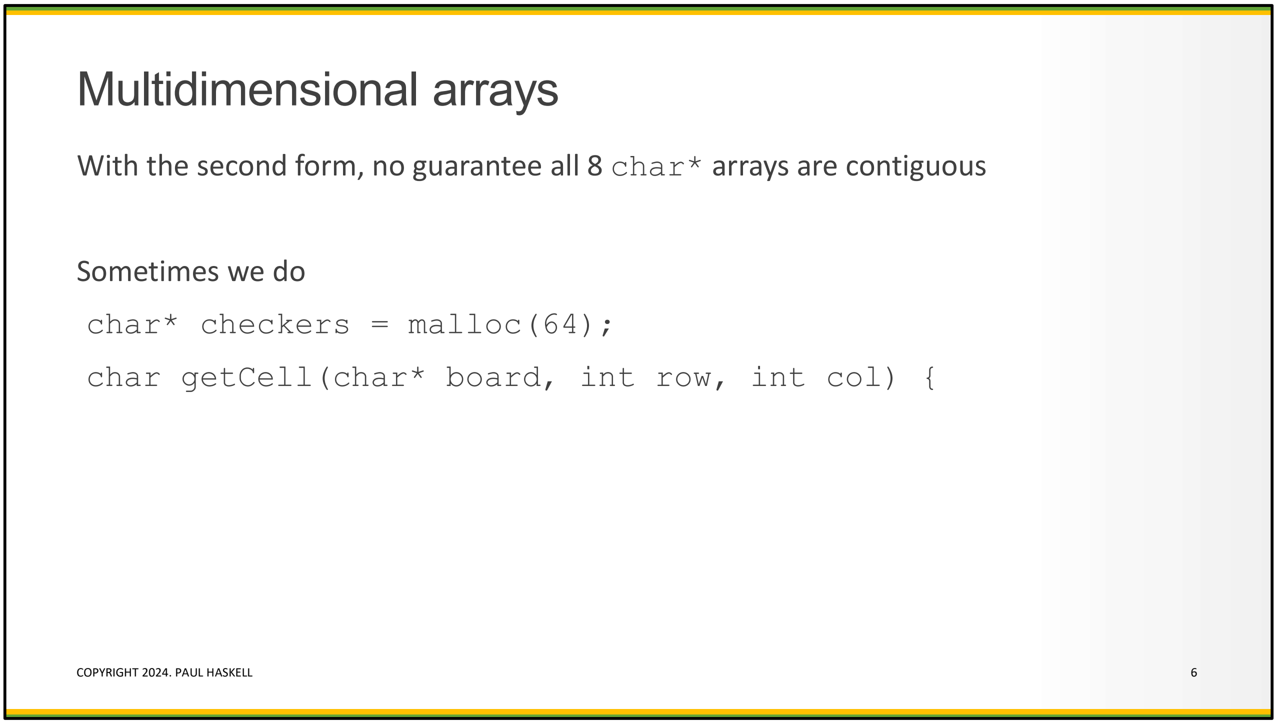
What is this?

Can we do more than two dimensions?

```
char hyperCheckers[8][8][8];
```

An array of 8 (arrays of 8 chars).
An array of 8 POINTERS TO chars. Need for(…) { checkers[i] = (char*) malloc(8); }

Pretty similar, huh? Second more work, but don't need to know sizes till runtime.



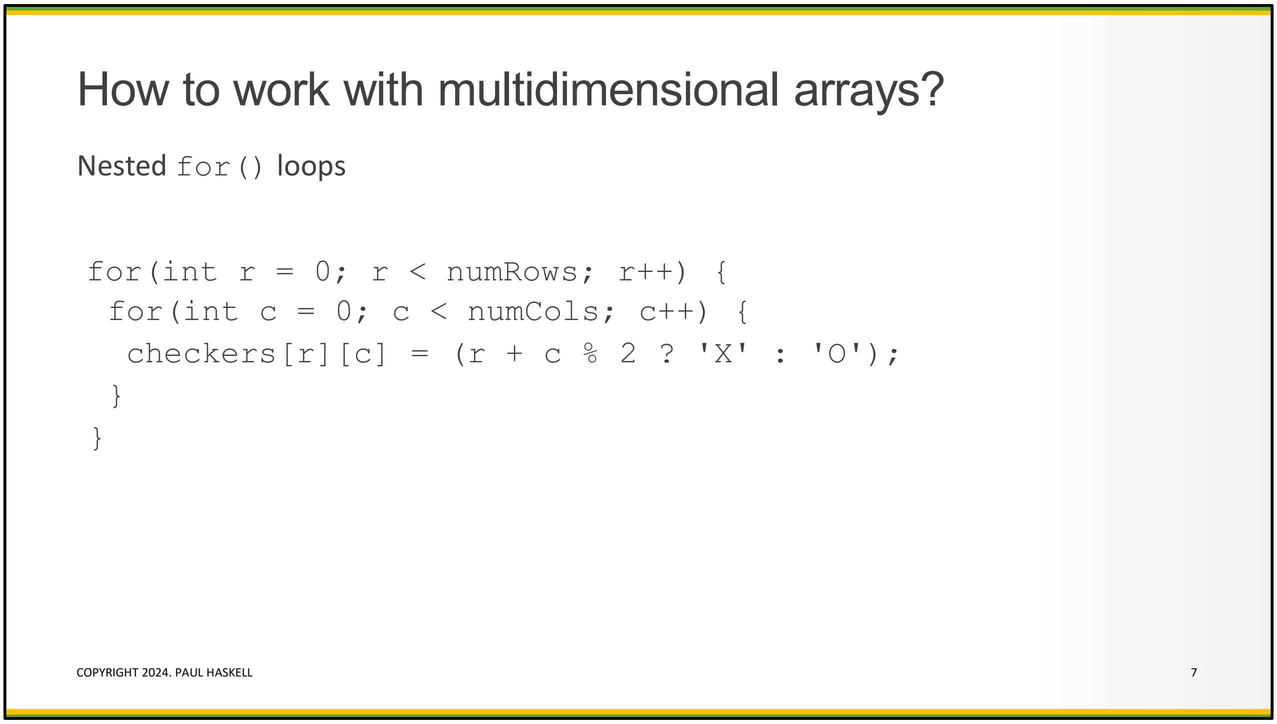
Multidimensional arrays

With the second form, no guarantee all 8 `char*` arrays are contiguous

Sometimes we do

```
char* checkers = malloc(64);
char getCell(char* board, int row, int col) {
```

What goes in the method?
return board[row*8+col]
Or
return board[row+8*col]

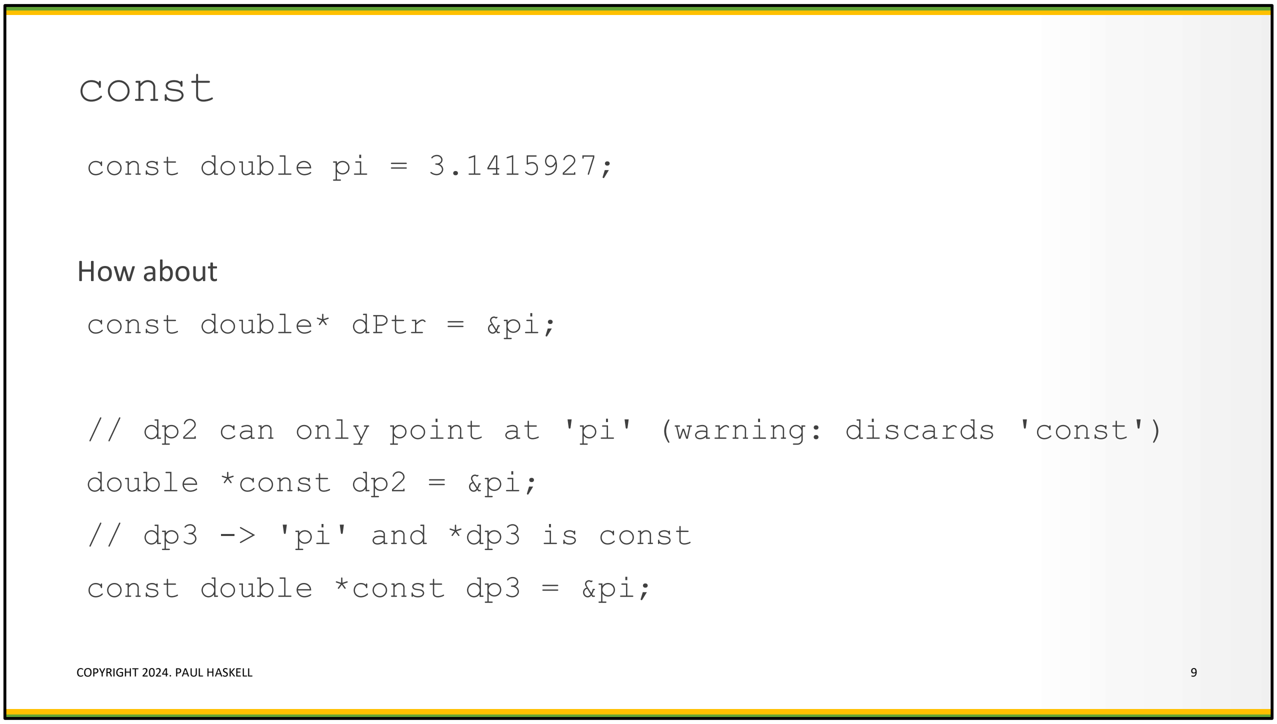


How to work with multidimensional arrays?

Nested `for()` loops

```
for(int r = 0; r < numRows; r++) {
 for(int c = 0; c < numCols; c++) {
  checkers[r][c] = (r + c % 2 ? 'X' : 'O');
 }
}
```

More new syntax



const

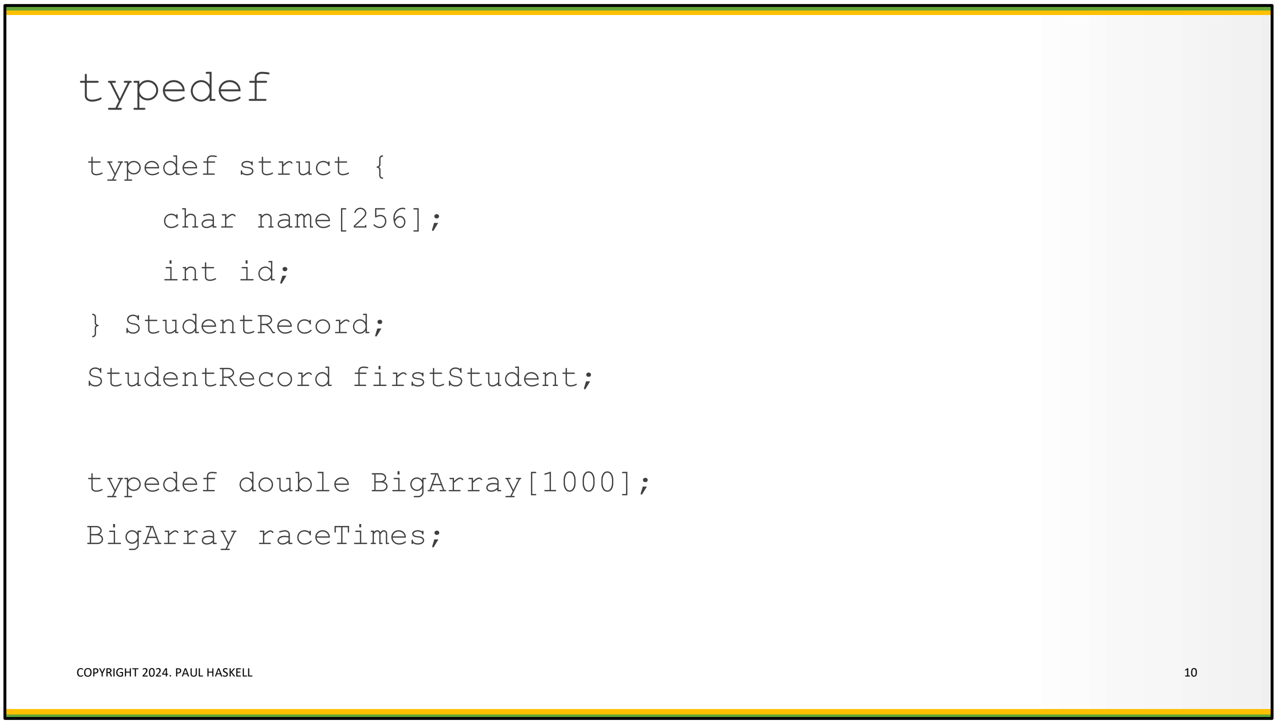
```
const double pi = 3.1415927;
```

How about

```
const double* dPtr = &pi;

// dp2 can only point at 'pi' (warning: discards 'const')
double *const dp2 = &pi;
// dp3 -> 'pi' and *dp3 is const
const double *const dp3 = &pi;
```

#1 pretty clear – just like Java's "final"
#2: Is the pointer 'const' or the double?
Read from RIGHT to LEFT: "a pointer to double that is const"
#3: const pointer to a double. Const pointer to a const double
See "constfun.c"

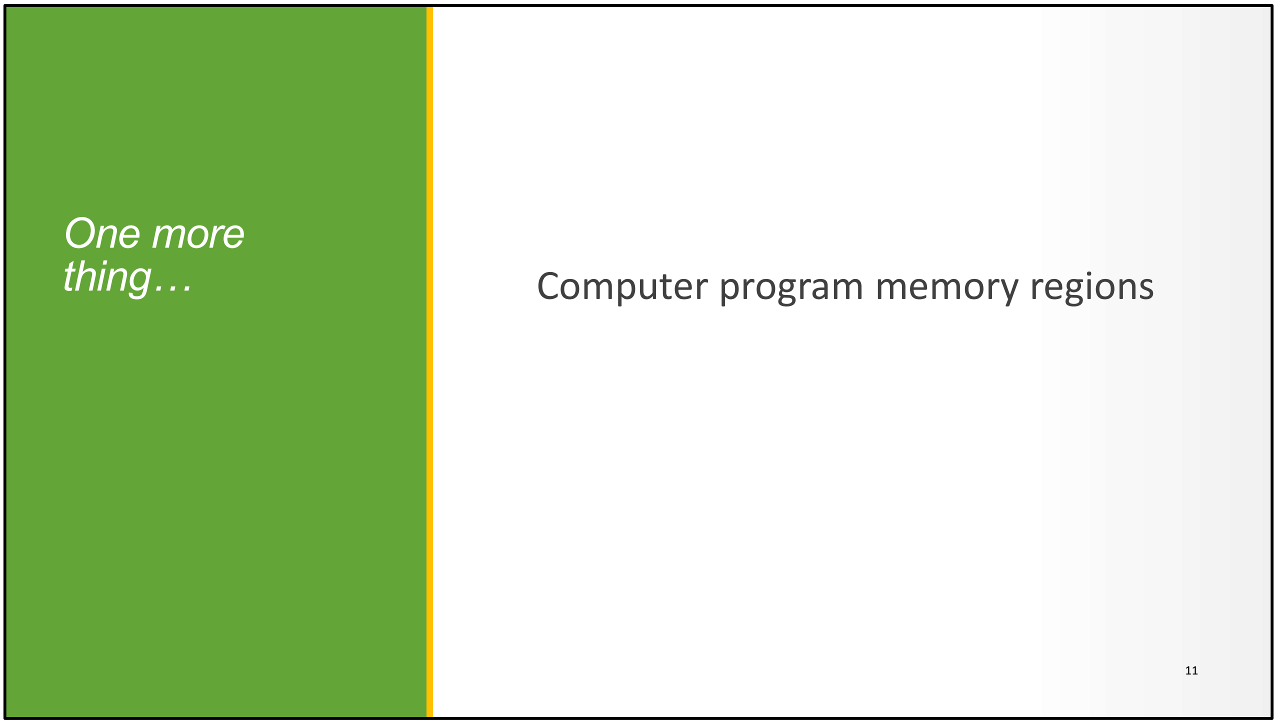


typedef

```
typedef struct {
    char name[256];
    int id;
} StudentRecord;
StudentRecord firstStudent;

typedef double BigArray[1000];
BigArray raceTimes;
```

typedef creates a new type, which can be used as if it were built in
Supposed variable name becomes the type's name



One more thing…

Computer program memory regions

11

Run AddrDemo.c several times

- Stack addrs > Heap addrs
- Not always the same
- Not actual physical addresses: virtual addresses for this program. Look at how big values are!

Boundary between Heap and Stack not fixed. Heap goes UP, Stack goes DOWN.

Hopefully they don't cross!