FCPA 2022  
  
Arrays

Student Workbook 06

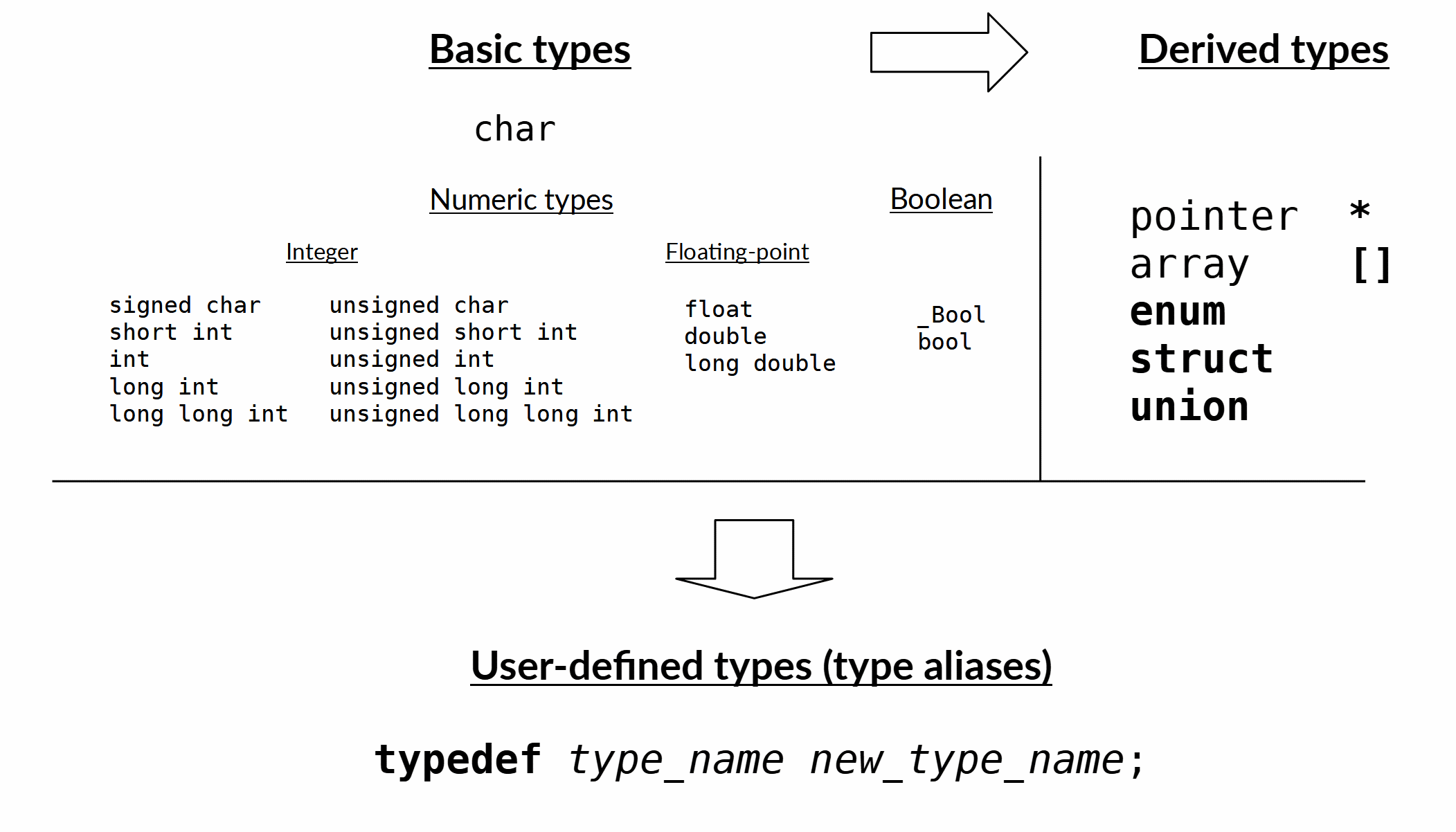
January 14, 2022

Author

Paul Kimball  
Interface Associates

1. Arrays

Data Types



* "bytes" and memory
  + Size and Alignment
* Character type
* Numeric types
  + Integral Types
  + Floating Point Types
* Boolean type
* Derived types

Arrays

* Often you'd like to use a list of related values
  + A day's worth of hourly temperature measurements
  + A shopping list
  + A table of values
* An array is a derived type
  + Holds multiple values (elements)
  + All elements are the same type
  + Contiguous block of memory
* Declare an array - this allocates memory

int lotto\_numbers[6];

* Access an array element with the [ ] operator
  + Integer indices are **zero-based**
  + Last index is size - 1

lotto\_numbers[0] = 19;

lotto\_numbers[1] = 12;

first = lotto\_numbers[0];

second = lotto\_numbers[1];

* Aggregate Initialization is simple

int lotto\_numbers[6] = {1,2,3,4,5,6};

// or, to automatically compute size

int lotto\_numbers[] = {1,2,3,4,5,6};

# For Loops

## The for loop is often used with arrays

for ( initializer; *test-expr*; *iterate-expr* ){

statement;

}

Example

int lotto\_numbers[6];

...

int size = sizeof(lotto\_numbers) / sizeof(int);

for (int i=0; i < size; i++)

printf("number %d is %d", i, lotto\_numbers[i]);

// or

for (int i=0; i < size; i++){

printf("number %d is %d", i, lotto\_numbers[i]);

}

# Arrays as pointers

* The *name* of an array *is a pointer to its first element*
  + Given the following declaration

**int** lotto\_numbers[6];

* + These are all the same address

lotto\_numbers

&lotto\_numbers[0]

&lotto\_number // I know, it's odd...

## And the type of the address in this case is "pointer to int"

**int \***

## Pointer arithmetic

* + When you add a number to a pointer, it's the same as changing the index in the [ ] operators
  + It moves the byte offset one *array element* farther along
  + These are the same value:

lotto\_numbers[4]

\*(lotto\_numbers + 4)

Passing an Array

* Arrays are (essentially) passed by reference
  + That is, the *value* that is copied to the local variable is the *pointer* to the array

**print\_number\_list(int \*numbers, int size) {**

for (int i = 0; i < size; i++) {

printf("%d\n", numbers[i]);

}

}

int main(void) {

int lotto\_nums[6] = { 23, 14, 12, 8, 6, 15 };

int size = sizeof(lotto\_nums)/sizeof (int);

// The address of the array is passed

**print\_number\_list(lotto\_nums, size);**

}

Advantages of Arrays

* Same identifier is used to access all members
* Easy to iterate through an array with a loop
* Compact in memory
* Passed by reference

Limitations of Arrays

* You must remember how big it is (!)
  + If you try to access past the last element, you "overflow your buffer"

If you're reading, you read garbage

If you're writing, you probably messed up the value of some other object

* No way to resize it (!)
  + If you need it bigger you must make a bigger one and copy all your old elements into it