FCPA 2022  
  
Arrays and Pointers

Student Workbook 06

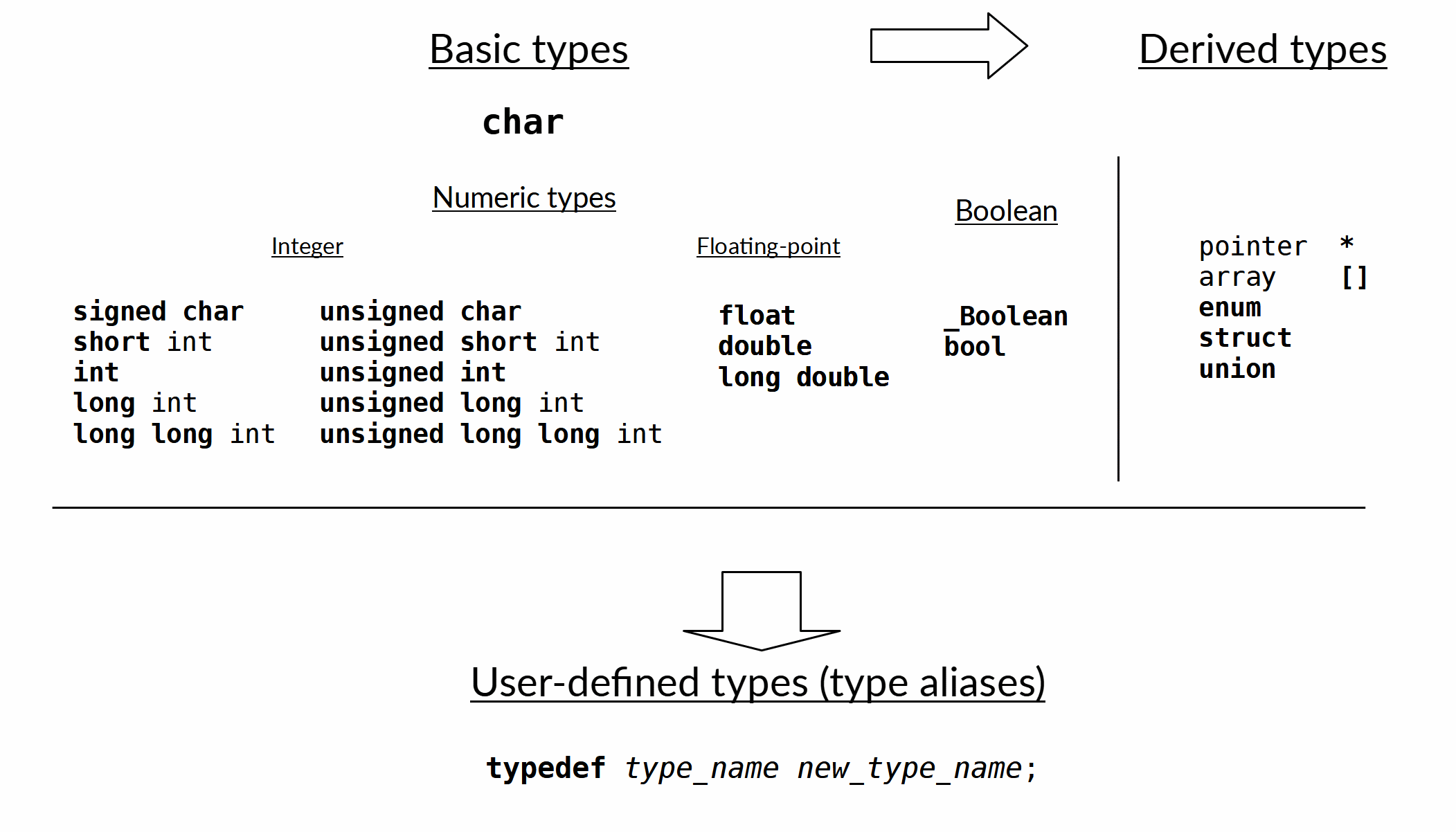
January 18, 2022

Author

Paul Kimball  
Interface Associates

1. Recap

Data Types



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

Identifiers

* An *identifier* is a name that the programmer gives to:
  + A variable
  + A function
  + A data type
* A *declaration* establishes an identifier and its important features

Data type

* Describes a kind of data that can be manipulated as a unit
  + One of the most important features of an identifier
* Some aspects of a data type:
  + Size and alignment in memory
  + Range and interpretation of binary values
  + Constant-ness
  + Storage location
  + Legal/illegal operations - what can you do with it?

Syntax

Semantics

Constraints

* + Relationship to other types
* The compiler uses data types to determine whether your code is legal, and how to convert it to machine code

Declarations and Types

* Data types are used in *ALL* declarations
* Variable declaration
  + Type describes the value(s) that a variable can hold

*data-type* ***variable-name***;

* Function declaration
  + Types declare what values a function accepts as parameters, and what it returns

*data-type* ***function-name(****data-type* ***param,*** *...****)****;*

* Data type declaration
  + Types declares how the new type is related to existing types
  + Tagged types

***enum tag-name******{*** *enumeration-list* ***}****;*

***struct tag-name******{*** *member-list* ***}****;*

***union tag-name******{*** *member-list* ***}****;*

* + Type alias

***typedef*** *old-type-name* ***new-type-name****;*

* Variables, functions, and data types must be declared before they are used in expressions

Symbols with Multiple Meanings

* Some symbols have slightly different meanings when they occur in a declaration vs an expression
* Equals sign
  + In a declaration, introduces an initializer value
  + In an expression, assigns a value to a variable
* Asterisk
  + In a declaration, indicates a derived pointer type
  + In an expression, dereferences a pointer to access its value
* Parentheses after an identifier
  + In a declaration, indicates a new function identifier
  + In an expression, calls an existing function
* Square brackets
  + In a declaration, indicates an array type
  + In an expression, accesses an element of an array

Logic and Flow of Control

* "Selection" statements
  + if / else
  + switch / case
* "Jump" statements
  + break
  + return
  + We haven't covered continue or goto yet
* "Iteration" statements
  + for
  + while
  + do while

Pointers and Arrays

* Pointers and arrays are related, but not identical
  + Both are derived types
  + Both can use pointer arithmetic or array indices in brackets
  + Both can be used to pass an array to a function by reference
* BUT...
* An array declaration allocates memory for all the array elements
  + You can initialize its values
* A pointer declaration allocates memory large enough for a single address
  + You can initialize it to the address of an existing object or function

How to Write a Program

* Think:
  + What do I want to do?
  + What have I got to work with?
  + How would I know if it works correctly?
* Describe the problem and solution in words
  + Make a drawing
  + Describe an algorithm
  + Tell someone else about it
* Get ready to write
  + Clear some space
  + Create a directory / VS project / Git repo
  + Open a file (.c)
* Write some code
  + Start with some comments to outline the task
  + Pick one small step
  + Write the simplest possible code to implement that step
* Build
  + Watch for warnings and errors - fix code
* Run it / Test it
  + Run every time you add a tiny bit of code
  + Try to break it; verify your assumptions
* Refactor it so it reads well to your future self
* Repeat until it's "good enough"