Textbook

- The material covered today is in chapter 9 except function pointers, chapter 7
- Please go through chapters 8 and 9 before the lab
Credits + feedback

- The quizzes in these slides are from http://www.cprogramming.com/

- The rest of the material is original, so feedback is welcome!
Structures & Arrays

- entry entries[10];
  int i;
  for (i=0; i<10; i++)
  {
    entries[i].count = 0;
  }

- ..nothing really special

- Structures can contain arrays as fields (see entry.word in an earlier example..)
Structures Containing Structures

- Example from the old textbook (*):

```c
struct time {
    int hour;
    int minute;
};

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

struct datetime{
    struct time t;
    struct date d;
}
```

(*) Kochan, S, "Programming in C"
Structures Containing Structures (cont.)

- struct datetime dt;
  dt.t.hour = 10;
  dt.d.day = 28;
Structures & Pointers

- Structures can contain pointers, just like any other variable type

- Pointers to structures deserve a bit more attention:
  ```
  entry entry1;
  entry * entryPtr;
  entryPtr = &entry1;
  entryPtr->frequency = 2;
  entryPtr->frequency++;  
  strcpy(entryPtr->word,"food");
  ```

- With structure pointers use "arrow" rather than "dot"
Structure Pointers & Functions

• It is normally a good idea to pass pointers to structures rather than structures

• Function arguments are copied: if a structure has many fields, function calls result in copying a lot of data (slow)

• Example:
  ```c
  void print_entry(const entry * arg)
  {
    printf("word: %s\n", arg->word);
    printf("freq: %d\n", arg->frequency);
    return;
  }
  ```
Quiz to check your understanding

• Which of the following is correct?
  - A. `myCar[name] = "ferrari";`
  - B. `strcpy(myCar[name], "ferrari");`
  - C. `strcpy(myCar.name, "ferrari");`
  - D. `myCar.name = "ferrari";`

• ..assuming:
```c
struct car{
    char name[32];
} myCar;
```
Quiz to check your understanding

• Which of the following is not correct?

  – A. `entryPtr->frequency++;`
  – B. `entryPtr->frequency = 44;`
  – C. `(*entryPtr).frequency = 44;`
  – D. `(*entryPtr)->frequency = 44;`

• ..assuming:
  `entry * entryPtr;`
  `entryPtr = & myEntry;`
  and myEntry being a structure with int field "frequency"
Working with generic data types

- In the "extra" part of the lab this week there is an example of working with generic data types
- This will involve generic pointers and pointers to functions
- We also need generic pointers to deal with memory allocation, one of the topics for next week
Generic Pointers: void *

- The keyword `void` can be used when declaring functions to mean "no arguments" or "no return value".

- The same keyword `void` can also be used to declare a *generic pointer*.

- A generic pointer is a pointer to variable of undefined type.

- Generally this is only useful as a transient format, to pass a pointer around functions without specifying the type.

- We need to cast a generic pointer to a specific pointer type before we can dereference it (i.e. use it).
More on generic pointers

• Any pointer can be casted to generic:
  
  ```
  int var = 5;
  int * ptr = &var;
  void * gen_ptr = (void *) ptr;
  ```

• However, before being able to do anything with the generic pointer we need to cast it back to a pointer of specific type
  
  ```
  int * another_ptr;
  another_ptr = (int *) gen_ptr;
  ```

• ..so if we always have to cast back, what's the point?
Pointers to functions

- Pointers to functions allow functions to be passed as arguments to another function
  - Sometimes referred to as callback functions
- Used a lot in real life (esp. with libraries) – examples:
  - Timers
  - GUI buttons (but also physical buttons when you work with micro controllers!)
  - Input validation (imagine an extended scanf)
  - And much much more..
Pointers to functions & generic pointers

• Pointers to functions can make code more reusable

• Sometimes it is convenient to let function pointers receive generic pointers as arguments

• Practical example: sorting
  – ..remember past lab?

• Note that the combination of function pointers and generic pointers does NOT happen all the time!
  Very often function pointers use specific type arguments
Dangers of generic pointers

• Once you have a generic pointer the compiler loses track of what the original type was

• Nothing prevents you from casting a generic pointer to a type different from the original: DISASTER!

• Be very careful about this kind of errors
Summary

- Structures can be used in combination with arrays and pointers
- Pointers to functions are used very often in real world situations, for example for callback functions
- Sometimes it is useful to use pointers to functions in combination with generic pointers (void *) – you will see a full example in the next lab