CS2014
Systems Programming

Lectures:
Stephen Farrell
stephen.farrell@cs.tcd.ie, x2354
Room: WR3.4 (impossible to find!)

Teaching Assistant:
Christian Cabrera
cabrerac@scss.tcd.ie, xNNNNN
See Christian @ labs
C Programming

• Learn by doing!
• Today: memory, structures, unions...
• If you’re interested in programming, play with doing the examples in some other programming language, e.g. python, PHP or whatever you prefer
  – Eventually, you’ll see the common concepts, the different concepts, and the syntactic sugar
  – But you don’t have to do that to pass the course or even to do well in the exam, C is enough for that
• Today: memory, structures, unions...
• Learn by doing! (Did I say that already?)
Layout of an array in memory

- From the perspective of your program, octets of an array are laid out in sequence.
- Special case: strings are arrays of chars
  - Strings are often null-terminated, i.e. have a zero valued octet at the end.
  - Some functions assume that, great source of bugs.

```c
char my_str[11];
snprintf(my_str,11,"foo");
```
Layout of two arrays in memory

- From the perspective of your program, two arrays on the stack are sequential
  - Not true of heap, when storage is allocated via `malloc()`

- DANGER!!! DANGER!!!
  - And somewhat fictional

```c
char my_str[7];
char my_other_str[3];
```

```
+-------------------------------+  +-------------------------------+
|   0   |   1   |   2   |   3   |  |   4   |   5   |   6   |   0   |
+-------------------------------+  +-------------------------------+
|   1   |   2   |     |     |  |     |     |     |     |
+---------------+                  +---------------+
```
Word size counts

• 8,16,32,64 bit processors differ
  – Endian-ness matters (and little-endian now dominant)

• Different OSes differ

• Portable C can work on all those if done well

• Bugs can manifest very differently

```c
int my_int;
char *my_other_str;
```

```
+-------------------+
|   i0  |   i1  |  i3   |  i3   |
+-------------------+
|   p0  |   p1  |  p2   |  p3   |
+-------------------+
```
Structured Data

- Lots of data is structured, e.g. account numbers and user information go together
- Struct in C allows you to handle that
- Similar to class concept in Java, C++ but limited to data, no code, no public/private
- Mostly used with typedef so struct can be used as a type, like int, char, ....

```c
// structure with some info about a file
typedef struct _finfo {
    char *name;
    char *link;
    char isdir;
} finfo, *finfo_p;

finfo myvar;
myvar.name="foo";
myvar.link=NULL;
myvar.isdir=0;
```
Layout in memory

- Items in a struct are laid out as they would be if they were on the stack (and defined in the same sequence and the compiler hasn’t optimised things)

```c
typedef struct _finfo {
    char *name;
    char *link;
    char isder;
} finfo, *finfo_p;
```
Unions

- Rarely used (we’ll see use in sockaddr later)
- Kind of like a choice
- Memory layout: enough octets to fit biggest element of union

```c
typedef union _ufinfo {
    char *name;
    char *link;
} ufinfo, *ufinfo_p;

ufinfo myvar;
myvar.name="foo";
```
C Programming Examples

• Ok, let’s go play with more examples