Numerical Types

- **int**: machine-dependent
- **Standard integers**
  - defined in stdint.h (#include <stdint.h>)
  - int8_t: 8-bits signed
  - int16_t: 16-bits signed
  - int32_t: 32-bits signed
  - int64_t: 64-bits signed
  - uint8_t, uint32_t, ...: unsigned
- **Floating point numbers**
  - float: 32-bits
  - double: 64-bits
Complex Types

- **Enumerations**
  (user-defined weekday: sunday, monday, ...)
- **Structures** (user-defined combinations of other types)
- **Unions** (same data, multiple interpretations)
- **Function pointers**
- **Arrays and Pointers of the above**
Enumerations

```c
enum days {mon, tue, wed, thu, fri, sat, sun};
// Same as:
// #define mon 0
// #define tue 1
// ...
// #define sun 6
```

```c
enum days {mon=3, tue=8, wed, thu, fri, sat, sun};
// Same as:
// #define mon 3
// #define tue 8
// ...
// #define sun 13
```
Enumerations

```c
enum days day;
// Same as:  int day;

for(day = mon; day <= sun; day++) {
    if (day == sun) {
        printf("Sun\n");
    } else {
        printf("day = %d\n", day);
    }
}
```
Enumerations

- Basically integers
- Can use in expressions like ints
- Makes code easier to read
- Cannot get string equiv.
- caution: `day++` will always add 1 even if enum values aren’t contiguous.
Structures

```c
struct mystruct {
    char name[32];
    int age;
    char *addr;
};
```
void foo(void) {
    struct mystruct person; // uninitialized

    struct mystruct person2 = {
        .name = {'f','o','o','\0'},
        .age = 22,
        .addr = NULL
    };

    // struct pointer
    struct mystruct *pptr =
        (struct mystruct *)malloc(sizeof(struct mystruct));

    ...
}
Structures

struct mystruct {
    char name[32];
    int age;
    char *addr;
};

...

person.age = 10;       // direct access
person.addr = (char *)malloc(64);

pptr->age = 24;        // indirect access
strncpy(pptr->name,"foo",32);   // through pointer
pptr->addr = NULL;

...
Structures

- Container for related data
- Chunks of memory; syntactic sugar for easy access.
- May have empty gaps between members
- Useful in creating data structures such as linked lists
Unions

union myunion {
    int x;
    struct {
        char b1;
        char b2;
        char b3;
        char b4;
    } b;
};

union myunion num;

num.x = 1000;
num.b.b1 = 5;
Unions

- Same memory space interpreted as multiple types
- Useful for plugins, slicing network packets etc.
Function Pointers

```c
int min(int a, int b);
int max(int a, int b);

int foo(int do_min) {
    int (*func)(int,int); // declaring func. ptr
    if (do_min)
        func = min;
    else
        func = max;

    return func(10,20); // indirect call
}
```
Function Pointers

- Points to a function
- Has a *-type of the function it points to
Renaming Types

- Complex types inconvenient to write over and over
  - (enum day *)malloc(sizeof(enum day))
  - (struct foo *)malloc(sizeof(struct foo))
  - (union bar *)malloc(sizeof(union bar))
  - (int (*)(int,int))((void *)min)

```c
typedef long old_type newtype
typedef enum day day_t;
typedef struct foo foo_t;
typedef int (fptr_t)(int,int);
```