Slide 1

Administrivia

- Reminder: Homework 8 due next week.

Slide 2

Minute Essay From Last Lecture

- Most people had not seen Conway's game before, though one had implemented it using OpenGL and one had heard it mentioned in a math course. (That it would be of interest in math I guess makes sense given that it was first published in a column called "Mathematical Games"?)
- One person thought it was interesting how simple rules could give such interesting results. Agreed!

Slide 3

User-Defined Types

- So far we've only talked about representing very simple types — numbers, characters, text strings, arrays, and pointers. You might ask whether there are ways to represent more complex objects, such as one can do with classes in object-oriented languages.
- The answer is "yes, sort of": C doesn't provide nearly as much syntactic help with object-oriented programming, but you can get something of the same effect. But first, some simpler user-defined types . . .

Slide 4

User-Defined Types in C — `typedef`

- `typedef` just provides a way to give a new name to an existing type, e.g.:

```
typedef charptr char *;
charptr c1, c2;
```

- Can help with readability: E.g., without `typedef` we'd write

```
char * c1, * c2;
```

and it's all too easy to forget that second `*`.

Slide 5

C `typedef`s, Continued

- `typedef` also useful to isolate things that might be different on different platforms (e.g., whether to use `float` or `double` in some application) in a single place, or where you want to easily be able to change a type used frequently.
- Example:

```
typedef myfloat double; /* change only this to use float */
myfloat f1;
myfloat f2;
```

Slide 6

User-Defined Types in C — `enum`

- In C (and in some other programming languages) an *enumeration* or an *enumerated type* provides a way to represent a variable that can be one of a list of values (e.g., a "color" that can be red, green, blue, or yellow).
- Two syntaxes (next slide) . . .

Slide 7

C enums, Continued

- One way is simple but a bit cumbersome:

```
enum basic_color_t { red, green, blue, yellow };
enum basic_color_t c = red; /* have to repeat "enum" */
```

- Another way uses `typedef`:

```
typedef enum { red, green, blue, yellow } basic_color_t;
basic_color_t c = red;
```

Slide 8

C enums, Continued

- Enumerated data types can make code more readable, and in C sometimes combine nicely with `switch` (to specify what happens for each value), e.g.,

```
enum basic_color_t c = /* something */
switch (c) {
    case red: /* something */
        break;
    case green: /* something */
        break;
    /* .... */
}
```

- But under the hood, C enumerated types are really just integers, and they can be ugly to work with in some ways (e.g., no nice way to do I/O with them).

Slide 9

User-Defined Types in C — `struct`

- More complex (interesting?) types can be defined with `struct`, which lets you define a new type as a collection of other types — something like a class in an object-oriented language, but with no methods and no way to hide fields/variables.
- Two versions of syntax (next slide) . . .

Slide 10

C `struct`s, Continued

- One way is simple but a bit cumbersome:

```
struct account_t {
    char acct_ID[9]; /* 8 characters plus '\0' */
    unsigned long balance;
};
struct account_t a;
```

- Another way uses `typedef`:

```
typedef struct {
    char acct_ID[9]; /* 8 characters plus '\0' */
    unsigned long balance;
} account_t;
account_t a1;
```

- Initialize field by field (next slide) or like this:

```
account_t a1 = { "12341234", 1000 };
```

Slide 11

C `struct`s, Continued

- Either way you define a `struct`, how you access its fields is the same:

  `.` if what you have is a `struct` itself:

  ```
  struct account_t acct;
  acct.balance = 0;
  ```

  `->` if what you have is a pointer to a `struct`:

  ```
  struct account_t a;
  struct account_t * acct_p = &a;
  acct_p->balance = 1;
  ```

  (could also use `(*acct_p).balance` but uglier?)

Slide 12

`struct`s, Continued

- (Look at example code briefly.)

Slide 13

User-Defined Types in C — `union`

- For completeness, we should mention that C also provides a way of defining a structure that can contain one of several alternatives ("this OR that", as opposed to the "this AND that" of `struct`) — `union`.
- For example, the following declares a data type that can hold either a `float` or an `int`:

```
union thing {
    float f;
    int i;
};
union thing t1;
```

  `t1` can hold either a `float` (`t1.f`) or an `int` (`t1.i`) but not both.
- More in textbook about this; it can be useful, but can also make code more difficult to understand.

Slide 14

User-Defined Types and Library Code

- Library code often makes use of "opaque" types (e.g., `FILE`).
- Implementing this often involves separating functionality into interface (`.h` file containing type definitions, function declarations) and implementation (`.c` file containing function definitions.
- This leads into . . .

Slide 15

Separate Compilation and `make` — Review

- C (like many languages) lets you split large programs into multiple source-code files. Typical to put function declarations (headers), constants, etc., in file ending `.h`, function definitions (code) in file ending `.c`. Compilation process can be separated into two steps: "compile" (convert source to object code) and "link" (combine object and library code to make executable).
- `make` can help manage compilation process. (Can also be useful as a convenient way to always compile with preferred options.)

Slide 16

Example — Sorted Linked List

- As an example, consider writing code for a sorted linked list.
- (You've probably seen something like this in another language, and the ideas are the same; it's just the details that are a little messier.)
- My example follows the scheme laid out in the previous slide:
  - a `.h` file that defines a type for the (nodes of) the list (we'll represent a list as a pointer to its first node) and declares some functions to perform operations on the list, and
  - a `.c` file with the code for the functions, and
  - additional files for a test program.
- (Code next time.)

Slide 17

Minute Essay

- Questions? otherwise just sign in.