Types

The type of an object determines the values it can have and the operations that can be performed on it.

Basic types:
- char: a character; typically a byte
- int: an integer; typically a word
- float: single-precision floating point
- double: double-precision floating point

Int qualifiers (optional):
- short int: "smaller" (not double precision)
- long int: "bigger" (but not double precision)

Unsigned integers: non-negative modulo n, where n is #bits/integer.
- unsigned int
- unsigned short int
- unsigned char

Is char signed or unsigned?

Types of Constants

- Character constant (single quote)
- Character code

Types

Type Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>7-8</td>
</tr>
<tr>
<td>short</td>
<td>16-32</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
</tr>
<tr>
<td>float</td>
<td>32-64</td>
</tr>
<tr>
<td>double</td>
<td>64-128</td>
</tr>
<tr>
<td>pointer</td>
<td>1-64</td>
</tr>
</tbody>
</table>

Note: C did not exist in 1964; this table just reflects typical sizes.
Arrays

- Arrays are declared by specifying the type and size. For example:
  ```
  int digits[10];
  ```
  - `digits` is an array of 10 integers.
  - Arrays may be indexed by any integer expression:
    ```
    digits[f(x)/2 + BASE]
    ```
  - No bounds checking!

- Multi-dimensional arrays:
  ```
  float matrix[3][4][5]
  ```
  - A 3-dimensional array with 15 elements.

Arrays are stored in row-major order, where the last subscript varies the fastest:
```
matrix[0][0][0], matrix[0][0][1], ...
```  

Strings & Initialization

- Strings are arrays of characters.
- The compiler always provides a terminating null character:
  ```
  "hello\n"
  ```

- Array length can be derived from initialization:
  ```
  char hello[] = "hello\n";
  ```
  ```
  is equivalent to
  ```
  char hello[7] = { 'h', 'e', 'l', 'l', 'o', '\n', 0 };
  ```

- Ditto for arrays:
  ```
  int x[] = { 1, 2, 3 };
  int y[][3] = {
    { 1, 3, 5 },
    { 2, 4, 6 },
    { 3, 5, 7 },
    { 4, 6, 8 }
  };
  ```
  - See K&R, sections 2.4 & 4.9 for more information.

Enumerations

- Enumerations associate constant values with identifiers:
  ```
  enum boolean { NO, YES }
  enum color { RED, GREEN, BLUE }
  ```

- Values are generated and may be printed symbolically by debuggers:
  ```
  enum escapes { BELL=\a, BACKSPACE=\b, TAB=\t, ... }
  enum months { Jan=1, Feb, Mar, Apr, May, June, Jul, ...
  ```

- enum identifiers should have no conflicts.

- Any declarations specify the number of elements, not the upper bound.

- What is the difference between enum and #define?
  ```
  ```

Strings & Initialization

- Strings are arrays of characters.