Types

CS 217

Types

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

• Base types
  char a character, typically a byte
  int an integer; typically a word
  float single-precision floating point
  double double-precision floating point

Type Qualifiers

• Length qualifiers
  short int (smaller; 16-bits on 32-bit machine)
  long int (larger; 32-bits on 32-bit machine)

• Unsigned integers
  unsigned int
  unsigned short int
  unsigned char

• Constant (read-only) variables
  const double pi = 3.14159
**Constant Expressions**

- Evaluated at compile time
  ```c
  int p = 1 - 1;
  ```
- Use constant expressions…
  - to reduce the number of `# define` constants
  - to increase readability
  - to improve changeability; e.g.,
  ```c
  # define MAXLINE
  --
  char buf[2*MAXLINE + 1];
  ```

**Types of Constants**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char 'a'</code></td>
<td>character constant (single quote)</td>
</tr>
<tr>
<td><code>'\035'</code></td>
<td>character code 35 octal</td>
</tr>
<tr>
<td><code>'\x29'</code></td>
<td>character code 29 hexadecimal</td>
</tr>
<tr>
<td><code>'t'</code></td>
<td>tab (<code>\011</code>, <code>do man ascii</code>)</td>
</tr>
<tr>
<td><code>'\n'</code></td>
<td>newline (<code>\012</code>)</td>
</tr>
<tr>
<td><code>null character</code></td>
<td></td>
</tr>
<tr>
<td><code>int 156</code></td>
<td>decimal constant</td>
</tr>
<tr>
<td><code>0234</code></td>
<td>octal</td>
</tr>
<tr>
<td><code>0xe</code></td>
<td>hexadecimal</td>
</tr>
<tr>
<td><code>long 156L</code></td>
<td></td>
</tr>
<tr>
<td><code>156L</code></td>
<td>don’t do it</td>
</tr>
<tr>
<td><code>float 15.6F</code></td>
<td></td>
</tr>
<tr>
<td><code>15.6f</code></td>
<td></td>
</tr>
<tr>
<td><code>double 15.6</code></td>
<td>defaults to <code>double</code></td>
</tr>
<tr>
<td><code>15.6L</code></td>
<td></td>
</tr>
<tr>
<td><code>15.6l</code></td>
<td></td>
</tr>
</tbody>
</table>

**Arrays**

- Array declarations specify the number of elements, not an upper bound on the index
  ```c
  int digits[10];
  ```
  says that `digits` is an array of 10 `int`
  ```c
  digits[0], digits[1], ... digits[9]
  ```
- Array may be indexed by integer expression
  ```c
  digits[f(x)/2 + BASE]
  ```
- No bounds checking!
Arrays (cont)

- Multi-dimensional arrays
  
  ```
  float matrix[3][4][5]
  ```
  
  is a 3-dimensional array w/ 3x4x5=60 elements

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

Strings & Initialization

- Arrays of characters
  
  ```
  "hello\n"
  ```
  
  the compiler always provides a terminating \0

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

Initialization (cont)

- Ditto for arrays...
  
  ```
  int x[] = { 1, 2, 3 };
  ```
  
  ```
  int y[1][3] = {
      { 1, 3, 5 },
      { 2, 4, 6 },
      { 3, 5, 7 },
      { 4, 6, 8 }
  };
  ```
Enumerations

- Associate constant values with identifiers
  ```
  enum boolean { FALSE, TRUE }; 
  enum color { RED, BLUE, GREEN }; 
  ```
- Enum identifiers are int constants
- Values can be specified
  ```
  enum escapes {TAB='\t', BACKSPACE='\b'}; 
  enum months {Jan=1, Feb, Mar, Apr, May, June, Jul, Aug, Sep, Oct, Nov, Dec}; 
  ```