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” `int`
  - `long int` “bigger” `int`, but **not** double precision

• Unsigned integers: non-negative modulo $2^n$ where $n$ is #bits/integer
  - `unsigned int` `unsigned short int` `unsigned char`

• Is `char` signed or unsigned?
## Type Sizes

<table>
<thead>
<tr>
<th>year</th>
<th>72–81</th>
<th>80–92</th>
<th>64–92</th>
<th>93–?</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>DEC-10</td>
<td>PCs</td>
<td>IBM360 VAX 68020 SPARC MIPS</td>
<td>R4000 DEC Alpha</td>
</tr>
<tr>
<td>char</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>short</td>
<td>18</td>
<td>16</td>
<td>16</td>
<td>16.32</td>
</tr>
<tr>
<td>int</td>
<td>36</td>
<td>16,32</td>
<td>32</td>
<td>32,64</td>
</tr>
<tr>
<td>type</td>
<td>long</td>
<td>36</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>float</td>
<td>36</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>double</td>
<td>72</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>pointer</td>
<td>18</td>
<td>16,32</td>
<td>32</td>
<td>64</td>
</tr>
</tbody>
</table>

Note: C did not exist in 1964; this table just reflects typical sizes
Types of Constants

char
'a'
character constant (single quote)
'\035'
character code 35 octal
'\x29'
character code 29 hexadecimal
't'
tab ('\011', do "man ascii" for details)
n'
newline ('\012')
'\'
backslash
'\'
single quote
'b'
backspace ('\010')
'0'
null character

int
156
decimal constant
0234
octal
0x9c
hexadecimal

long
156L
156l for sanity, use upper-case L

float
15.6f
1.56e1F

double
15.6
“plain” floating point constants are doubles
15.6L
15.6l
Constant Expressions

- **Const** qualifier identifies *read-only variables*
  
  ```
  const double Pi = 3.14159;
  const double TwoPi = 2*3.14159;
  ```

- **Constant expressions** are evaluated at *compile time*
  
  ```
  int p = 1 - 1;
  int p = 1/0, x = 1 ? 0 : 1/0;
  ```

- Use constant expressions to reduce the number of *define* constants to increase readability to improve changeability, e.g.
  
  ```
  #define MAXLINE 120
  ...
  char buf[2*MAXLINE + 1];
  ```
Arrays

- Array declarations specify the number of elements, not the upper bound

```java
int digits[10];

digits is an array of 10 ints
digits[0], digits[1], ..., digits[9]
```

- Arrays may be indexed by any integer expression

```java
digits[f(x)/2 + BASE]
```

- No bounds checking!

- Multi-dimensional arrays

```java
float matrix[3][4][5]

a 3-dimensional array with 3 \times 4 \times 5 = 60 elements
```

- Arrays are stored in row-major order; last subscript varies “fastest”

```java
matrix[0][0][0], matrix[0][0][1], ...
```
Strings & Initialization

"Strings" are arrays of characters

"hello\n"

the compiler always provides a terminating '\0'

Array length can be derived from initialization

char hello[] = "hello\n";

is equivalent to

char hello[7] = "hello\n";
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 }
};

will be 4 — number of 3-element rows

these braces can be omitted

see K&R, sections 2.4 & 4.9 for more information
Enumerations

- **Enumerations** associate constant values with identifiers

  ```c
  enum boolean { NO, YES };
  enum color { RED, GREEN, BLUE };
  ```

- Values are generated and may be printed symbolically by debuggers

- Values can be given and unspecified ones automatically continue

  ```c
  enum escapes { BELL='\a', BACKSPACE='\b', TAB='\t'};
  enum months { Jan=1, Feb, Mar, Apr, May, Jun, Jul,
      Aug, Sep, Oct, Nov, Dec };
  ```

- **enum** identifiers are `int` constants, but enumeration type may take less space

  ```c
  sizeof NO is 4 bytes
  ```

  ```c
  enum boolean flag; may occupy 1–4 bytes
  ```

- **enum** identifiers should have no *conflicts*

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