Types

What is a type?

A *type* is a collection of values and a collection of operations on those values.

Why does a language need types?

To determine how a sequence of bits is to be interpreted by the compiler.

<table>
<thead>
<tr>
<th>..., -2, -1, 0, 1, 2, ...</th>
<th>+, -, x, /, &gt;, &lt;, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>true, false</td>
<td>∨, ∧, ¬</td>
</tr>
</tbody>
</table>

0100 0000 0101 1000 0000 0000 0000 0000
Static and Dynamic Typing

- A language is **statically typed**, if the types of all variables are determined or specified at compile time
  - Static typing catches bugs earlier

- A language is **dynamically typed** if the type of a variable can vary at run time depending on the value assigned
  - Dynamic typing makes coding easier
Type Errors

✦ A type error is when a program executes an operation on a data type for which the operation is undefined.

✦ A type system can provide error detection.

A type system is a precise definition of the bindings between the type of a variable, its values, and the possible operations on those values.

✦ A programming language is strongly typed if its type system allows all type errors in a program to be detected either at compile time or at run time.

• Tradeoff: reliability vs. flexibility.
Basic Types

- Supported by contemporary machines
- Standard for floating point data
- Integer formats and manipulations are not common

<table>
<thead>
<tr>
<th>Type</th>
<th>C</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>short, int, long</td>
<td>byte</td>
</tr>
<tr>
<td>Integer</td>
<td>short, int, long</td>
<td>short, int, long</td>
</tr>
<tr>
<td>Real Number</td>
<td>float, double</td>
<td>float, double</td>
</tr>
<tr>
<td>Character</td>
<td>char</td>
<td>char</td>
</tr>
<tr>
<td>Boolean</td>
<td>boolean</td>
<td></td>
</tr>
</tbody>
</table>
Formats

- Java defines data type sizes explicitly
- C lets the compiler define the format based on machine hardware
- Tradeoff: portability vs. optimization
- Organization of bits and bytes within a data type is machine dependent
  - Big endian, small endian

<table>
<thead>
<tr>
<th>Type</th>
<th>C</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1B</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>1B</td>
<td>2B</td>
</tr>
<tr>
<td>short</td>
<td>2B</td>
<td>2B</td>
</tr>
<tr>
<td>int</td>
<td>short &lt;= int &lt;= long</td>
<td>4B</td>
</tr>
<tr>
<td>long</td>
<td>&gt;= int and &gt;= 4B</td>
<td>8B</td>
</tr>
<tr>
<td>float</td>
<td>IEEE 754: 4B</td>
<td>IEEE 754: 4B</td>
</tr>
<tr>
<td>double</td>
<td>IEEE 754: &gt;= 8B</td>
<td>IEEE 754: 8B</td>
</tr>
</tbody>
</table>
Example: Case Conversion

```c
#include <stdio.h>

/* lower: convert c to lower case; ASCII only */
int lower (int c) {
    if (c >= 'A' && c <= 'Z')
        return c - 'A' + 'a';
    else
        return c;
}

int main (int arg, char *argv[]) {
    char ca = 'A';
    int i = 0x4A;
    char la = lower('A');
    char lj = lower(0x4A);

    printf("lower case of %c is %c \n", ca, la);
    printf("lower case of %c is %c \n", i, lj);
}
```

lower case of A is a
lower case of J is j
Type Conversions

- Both operands must be the same type in arithmetic operations

- Implicit conversions: automatic conversions
  - A set of implicit conversion rules (e.g. `int ~> float`)

- Explicit conversion: type casting
  - `(type name) expression` (e.g. `(int)3.14`)

- May loss information