What are types?

- Data types are sets of values along with operations that manipulate them.
- Values must be mapped to data types provided by the hardware and operations compiled to sequences of hardware instructions.
- Example: integers in C are made up of the set of values ..., -1, 0, 1, 2, ... along with operations such as addition, subtraction, multiplication, division...
Types representation

- **Basic types**
  - `int` - used for integer numbers
  - `float` - used for floating point numbers
  - `double` - used for large floating point numbers
  - `char` - used for characters
  - `void` - used for functions without parameters or return value
  - `enum` - used for enumerations

- **Composite types**
  - pointers to other types
  - functions with arguments types and a return type
  - arrays of other types
  - structs with fields of other types
  - unions of several types
Qualifiers, modifiers, storage

- **Type qualifiers**
  - *short* - decrease storage size
  - *long* - increase storage size
  - *signed* - request signed representation
  - *unsigned* - request unsigned representation

- **Type modifiers**
  - *volatile* - value may change without being written to by the program
  - *const* - value not expected to change

- **Storage class**
  - *static* - variable that are global to the program
  - *extern* - variables that are declared in another file
# Sizes

<table>
<thead>
<tr>
<th>Type</th>
<th>Range (32-bits)</th>
<th>Size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed char</td>
<td>-128 to +127</td>
<td>1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>0 to +255</td>
<td>1</td>
</tr>
<tr>
<td>signed short int</td>
<td>-32768 to +32767</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>0 to +65535</td>
<td>2</td>
</tr>
<tr>
<td>signed int</td>
<td>-2147483648 to +2147483647</td>
<td>4</td>
</tr>
<tr>
<td>unsigned int</td>
<td>0 to +4294967295</td>
<td>4</td>
</tr>
<tr>
<td>signed long int</td>
<td>-2147483648 to +2147483647</td>
<td>4 or 8</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>0 to +4294967295</td>
<td>4 or 8</td>
</tr>
<tr>
<td>signed long long int</td>
<td>-9223372036854775808 to +9223372036854775807</td>
<td>8</td>
</tr>
<tr>
<td>unsigned long long int</td>
<td>0 to +18446744073709551615</td>
<td>8</td>
</tr>
<tr>
<td>Float</td>
<td>$1 \times 10^{-37}$ to $1 \times 10^{37}$</td>
<td>4</td>
</tr>
<tr>
<td>Double</td>
<td>$1 \times 10^{-308}$ to $1 \times 10^{308}$</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>$1 \times 10^{-308}$ to $1 \times 10^{308}$</td>
<td>8, 12, or 16</td>
</tr>
</tbody>
</table>

`sizeof(x)` returns the size in bytes.
Characters representation

- ASCII code (American Standard Code for Information Interchange): defines 128 character codes (from 0 to 127),
- In addition to the 128 standard ASCII codes there are other 128 that are known as extended ASCII, and that are platform-dependent.
- Examples:
  The code for ‘A’ is 65
  The code for ‘a’ is 97
  The code for ‘0’ is 48
Understanding types in C matters …

- Incorrect use may result in bugs
  - There are implicit conversions that take place and they may result in truncation
  - Some data types are not interpreted the same way on different platforms, they are machine-dependent
    - `sizeof(x)` returns the size in bytes of the object `x` (either a variable or a type) on the current architecture
- Ineffective use may result in higher cost
  - Storage, performance
What will this program output?

```c
#include <stdio.h>
int main() {
    char c = -5;
    unsigned char uc = -5;

    printf("%d  %d \n", c, uc);

    return 0;
}
```
Printf format

c Character

d or i Signed decimal integer

f Decimal floating point

s String of characters

u Unsigned decimal integer

x Unsigned hexadecimal integer

p Pointer address

NOTE: read printf man pages for additional formats
What will this program output?

```
#include <stdio.h>
int main() {
    char c = 'a';

    printf("%c %d %x \n", c, c, c);

    return 0;
}
```
Pointers

- The address of a location in memory is also a type based on what is stored at that memory location
  - `char *` is “a pointer to char” or the address of memory where a char is stored
  - `int *` points to a location in memory where a int is stored
  - `float *` points to a location in memory where a float is stored
- We can do operations with this addresses
- The size of an address is platform dependent.
& and *

- Given a variable v
  \&v means the address of v

- Given a pointer ptr
  \*ptr means the value stored at the address specified by ptr