Statements and expressions

C has only a few basic types:

`int` An integer (`0`, `3`, `-14`)

`char` A one-byte integer suitable for ASCII characters (`'a'`, `'\t'`, `'3'`)

`float` A real number (`3.14159`, `-2.3`, `5.3e8`, `0.002`)

All C types have a fixed size in bytes, which is system independent; if you ever need to know the size of a type, you can use the `sizeof` operator.

sizeof.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
        printf("Integers are %zu bytes.\n", sizeof(int));
        return 0;
}
```

On my system, as on many, integers are 32 bits, or 4 bytes.

```
$ gcc -Wall -g -o sizeof sizeof.c
$ ./sizeof
Integers are 4 bytes.
```

Naturally, fixed sizes implies a fixed number of representable values. On a system with 32-bit integers, the largest possible `int` is `2147483647` and the smallest is `-2147483648`. The following program tries to compute a number that can not be represented in an `int`.

overflow.c

```
#include <stdio.h>

int main(int argc, char **argv)
{
        int big = 2147483647;
        printf("%d + 1 = %d\n", big, big + 1);
        return 0;
}
```

The result is overflow; the numbers simply wrap around.

```
$ gcc -Wall -g -o overflow overflow.c
$ ./overflow
2147483647 + 1 = -2147483648
```

C also has type modifiers for length, `short` and `long`. A `long int` is generally twice the size of an `int`, while a `short int` is generally half. There is also a `long long int`.

Actually, `char` is also an integer type, and it is always one byte. Although its name suggests it is only for storing characters (which in ASCII are always one byte), it is actually a fully-fledged signed integer type.

Integer types can also be explicitly `signed` or `unsigned`. With neither modifier, the default is `signed`, so all of the code so far would be identical if `int` everywhere had actually been typed `signed int`. An `unsigned int` has the same size as an `int`, but can not represent negative numbers, so its positive range is about twice as large. Overflowing an `unsigned` type wraps back to zero.

A `float` stores real numbers, with both a mantissa (fractional part) and exponent, so it can represent both very large and very small values. Nonetheless, it, too, has fixed size and thus fixed range; there are a few special `float` values to explicitly represent overflow (as positive or negative infinity) and the result of dividing by zero (NaN, or "not a number"). If you need more range than a `float`, a `double` has, unsurprisingly, double the space. Some systems also have a `long double`.

There is a header file, `limits.h`, which contains symbolic constants for the minimum and maximum values of the various integer types. The following program will employ `limits.h` and `sizeof` to describe the various C types as implemented on your system.

`sizeof2.c`

```c
#include <stdio.h>
#include <limits.h>

int main(int argc, char **argv)
{
        printf("char (%zu bytes)\n", sizeof(char));
        printf("  signed char %hhd to %hhd\n", SCHAR_MIN, SCHAR_MAX);
        printf("  unsigned char 0 to %hhu\n", UCHAR_MAX);
        printf("short int (%zu bytes)\n", sizeof(short int));
        printf("  signed short int %hd to %hd\n", SHRT_MIN, SHRT_MAX);
        printf("  unsigned short int 0 to %hu\n", USHRT_MAX);
        printf("int (%zu bytes)\n", sizeof(int));
        printf("  signed int %d to %d\n", INT_MIN, INT_MAX);
        printf("  unsigned int 0 to %u\n", UINT_MAX);
        printf("long int (%zu bytes)\n", sizeof(long int));
        printf("  signed long int %ld to %ld\n", LONG_MIN, LONG_MAX);
        printf("  unsigned long int 0 to %lu\n", ULONG_MAX);
        printf("long long int (%zu bytes)\n", sizeof(long long int));
        printf("  signed long long int %lld to %lld\n", LLONG_MIN, LLONG_MAX);
        printf("  unsigned long long int 0 to %llu\n", ULLONG_MAX);
        printf("float (%zu bytes)\n", sizeof(float));
        printf("double (%zu bytes)\n", sizeof(double));
        printf("long double (%zu bytes)\n", sizeof(long double));
        return 0;
}
```

An on my system, the result is:

```
$ gcc -Wall -g -o sizeof2 sizeof2.c
$ ./sizeof2
char (1 bytes)
  signed char -128 to 127
  unsigned char 0 to 255
short int (2 bytes)
  signed short int -32768 to 32767
  unsigned short int 0 to 65535
int (4 bytes)
  signed int -2147483648 to 2147483647
  unsigned int 0 to 4294967295
long int (4 bytes)
  signed long int -2147483648 to 2147483647
  unsigned long int 0 to 4294967295
long long int (8 bytes)
  signed long long int -9223372036854775808 to 9223372036854775807
  unsigned long long int 0 to 18446744073709551615
float (4 bytes)
double (8 bytes)
long double (12 bytes)
```

The `sizeof2.c` source code also contains a wealth of `printf` format string examples for various types. The integer types use the format character `d` (`signed`) or `u` (`unsigned`), perhaps prefixed with a length `hh` (`char`), `h` (`short`), `l` (`long`) or `ll` (`long long`). (You might also catch the length `z`, for the special type `size_t`, which is the return type of `sizeof`. It's a long story.) To print a `float` or `double`, incidentally, use the format character `f`.

Declarations and definitions Math expressions + - * / etc. Assignment Simple statements Block statements If Boolean tests While and Do-while For Switch