6 Various types of computer data

Machine instructions generally manipulate data stored in central memory. Data is organised as follows (there is some repetition here).

- The fundamental unit of data is a *bit*, something which can have two values, 0 or 1. Central memory is a very large collection of bits, possibly billions.
- Before the 1970s central memory was composed of many (about a million) small doughnut-shaped magnets threaded together with copper wire and called *magnetic core memory.* Hence the word 'core' used to mean central memory, and 'core dump' for a display of the contents of central memory (usually following a program crash). Nowadays, billions of bits of memory are stored on a single chip.
- Bits are never read singly. Memory is grouped into 8-bit units called *bytes.* Each byte then can have $2^8 = 256$ values. A byte then corresponds to a number in the range 0 (00000000) to 255 (11111111).
- The ascii character set maps all printable characters, such as 0, a, &, *, to byte values. Also, nonprintable characters such a carriage return, backspace, ctrl-U, etcetera (ctrl-G is 07 in Hex. It should make a sound when pressed — or printed).
- As far as I know, the smallest piece of data in C is a single byte, and the keyword is `char` because of the ascii conventions. In other words, when you need to present data byte-by-byte in a C program, you will use the word `char`.
- Next is `short` (short integer). In our system this appears to be two bytes with 65536 different values.

  In the 1990s the default integer length was 16 bits (short). Now that memory is much more abundant, the default is 32 bits.
- Next is `int` (integer), 32 bits. The range is from -2147483648 to 2147483647. About ± 2 billion.
- Next is `long`. On 32-bit machines this appears to be 4 bytes, on 64-bit machines this is 8 bytes.
- Memory addresses are important in C. There is no special keyword for 'memory address' — they are introduced in another way — but all memory addresses occupy 4 bytes or 8 bytes. On 32-bit machines the range is $0 \ldots 2^{32} - 1$. The highest memory address is 4294967295, 4 gigabytes.
- Next is `float`. In our system this appears to be a 4-byte representation of floating-point numbers.
- Next is `double`. In our system this appears to be a 8-byte representation of floating-point numbers.

The following program shows the size (number of bytes) in each data type. **sizeof(...)** gives the size in bytes of a data type. **sizeof(char *)** brings in an advanced C feature: **(char *)** means an address to data of type **char**. It will be introduced roughly halfway through the term.

```
#include <stdio.h>

main()
{

  printf("char %d bytes\n", sizeof(char));
  printf("short %d bytes\n", sizeof(short));
  printf("int %d bytes\n", sizeof(int));
  printf("float %d bytes\n", sizeof(float));
  printf("long %d bytes\n", sizeof(long));
  printf("double %d bytes\n", sizeof(double));

        /*
         * Working with addresses is an advanced topic.
         * Just to give a foretaste,
         * ‘short *’ means ‘address of a short integer,’,
         * ‘int *’ means ‘address of an ‘int’’, and
         * so on.  All these addresses are 4 or 8 bytes,
         * depending on the machine.
         */


  printf("address of char %d bytes\n", sizeof(char * ));
  printf("address of short %d bytes\n", sizeof(short * ));
  printf("address of int %d bytes\n", sizeof(int * ));
  printf("address of float %d bytes\n", sizeof(float * ));
  printf("address of long %d bytes\n", sizeof(long * ));
  printf("address of double %d bytes\n", sizeof(double * ));
}

Output when run on my 32-bit office PC:

char 1 bytes
short 2 bytes
int 4 bytes
float 4 bytes
long 4 bytes
double 8 bytes
address of char 4 bytes
address of short 4 bytes
address of int 4 bytes
```

```
address of float 4 bytes
address of long 4 bytes
address of double 4 bytes

Output when run on the 64-bit machine aturing:

char 1 bytes
short 2 bytes
int 4 bytes
float 4 bytes
long 8 bytes
double 8 bytes
address of char 8 bytes
address of short 8 bytes
address of int 8 bytes
address of float 8 bytes
address of long 8 bytes
address of double 8 bytes
```

Here are the internal representations of various numbers (Most of them need explanation)

```
            char z:  7a 00 00 00 00 00 00 00 00 00
          short 43:  2b 00 00 00 00 00 00 00 00 00
         short -43:  d5 ff 00 00 00 00 00 00 00 00
          short -9:  f7 ff 00 00 00 00 00 00 00 00
      short -32768:  00 80 00 00 00 00 00 00 00 00
       short 32767:  ff 7f 00 00 00 00 00 00 00 00
            int -2:  fe ff ff ff 00 00 00 00 00 00
           int 300:  2c 01 00 00 00 00 00 00 00 00
          int -300:  d4 fe ff ff 00 00 00 00 00 00
         int 70000:  70 11 01 00 00 00 00 00 00 00
        int -70000:  90 ee fe ff 00 00 00 00 00 00
   int -2147483648:  00 00 00 80 00 00 00 00 00 00
    int 2147483647:  ff ff ff 7f 00 00 00 00 00 00
           long -3:  fd ff ff ff 00 00 00 00 00 00
 float 1234.560059:  ec 51 9a 44 00 00 00 00 00 00
double 1234.560000:  0a d7 a3 70 3d 4a 93 40 00 00
      string hello:  68 65 6c 6c 6f 00 00 00 00 00
```