CHAPTER 2

Number Formats: The Representation of Large Numbers in C

> So I have made up my own system for writing large numbers and I am going to use this chapter as a chance to explain it
>
> —Isaac Asimov, *Adding a Dimension*

> The process that has led to the higher organization of this form could also be imagined differently
>
> —J. Weber, *Form, Motion, Color*

ONE OF THE FIRST STEPS in creating a function library for calculating with large numbers is to determine how large numbers are to represented in the computer's main memory. It is necessary to plan carefully, since decisions made at this point will be difficult to revise at a later time. Changes to the internal structure of a software library are always possible, but the user interface should be kept as stable as possible in the sense of "upward compatibility."

It is necessary to determine the order of magnitude of the numbers to be processed and the data type to be used for coding these numerical values.

The basic function of all routines in the FLINT/C library is the processing of natural numbers of several hundred digits, which far exceeds the capacity of standard data types. We thus require a logical ordering of a computer's memory units by means of which large numbers can be expressed and operated on. In this regard one might imagine structures that automatically create sufficient space for the values to be represented, but no more than is actually needed. One would like to maintain such economically organized housekeeping with respect to main memory by means of dynamic memory management for large numbers that allocates or releases memory according to need in the course of arithmetic operations. Although such can certainly be realized (see, for example, [Skal]), memory management has a price in computation time, for which reason the

representation of integers in the FLINT/C package gives preference to the simpler definition of static length.

For representing large natural numbers one might use vectors whose elements are a standard data type. For reasons of efficiency an unsigned data type is to be preferred, which allows the results of arithmetic operations to be stored in this type without loss as `unsigned long` (defined in `flint.h` as `ULONG`), which is the largest arithmetic standard C data type (see [Harb], Section 5.1.1). A `ULONG` variable can usually be represented exactly with a complete register word of the CPU.

Our goal is that operations on large numbers be reducible by the compiler as directly as possible to the register arithmetic of the CPU, for those are the parts that the computer calculates "in its head," so to speak. For the FLINT/C package the representation of large integers is therefore by means of the type `unsigned short int` (in the sequel `USHORT`). We assume that the type `USHORT` is represented by 16 bits and that the type `ULONG` can fully accept results of arithmetic operations with `USHORT` types, which is to say that the informally formulated size relationship $\texttt{USHORT} \times \texttt{USHORT} \leq \texttt{ULONG}$ holds.

Whether these assumptions hold for a particular compiler can be deduced from the ISO header file `limits.h` (cf. [Harb], Sections 2.7.1 and 5.1). For example, in the file `limits.h` for the GNU C/C++ compiler (cf. [Stlm]) the following appears:

```
#define UCHAR_MAX 0xffU
#define USHRT_MAX 0xffffU
#define UINT_MAX 0xffffffffU
#define ULONG_MAX 0xffffffffUL
```

One should note that with respect to the number of binary places there are actually only three sizes that are distinguished. The type `USHRT` (respectively `USHORT` in our notation) can be represented in a 16-bit register; the type `ULONG` fills the word length of a CPU with 32-bit registers. The type `ULONG_MAX` determines the value of the largest unsigned whole numbers representable by scalar types (cf. [Harb], page 110).[1] The value of the product of two numbers of type `USHRT` is at most `0xffff * 0xffff` = `0xfffe0001` and is thus representable by a ULONG type, where the least-significant 16 bits, in our example the value 0x0001, can be isolated by a cast operation into the type `USHRT`. The implementation of the basic arithmetic functions of the FLINT/C package is based on the above-discussed size relationship between the types `USHORT` and `ULONG`.

An analogous approach, one that used data types with 32-bit and 64-bit lengths in the role of `USHORT` and `ULONG` in the present implementation, would reduce the calculation time for multiplication, division, and exponentiation

[1] Without taking into account such practical nonstandard types as `unsigned long long` in GNU C/C++ and certain other C compilers.