CHAPTER 15

The LINT Public Interface: Members and Friends

> Please accept my resignation. I don't want to belong to any club that will accept me as a member.
>
> —Groucho Marx

> Every time I paint a portrait I lose a friend
>
> —John Singer Sargent

IN ADDITION TO THE CONSTRUCTOR functions and operators already discussed, there exist further LINT functions that make the C functions developed in Part I available to LINT objects. In the following discussion we make a rough separation of the functions into the categories "arithmetic" and "number-theoretic." The implementation of the functions will be discussed together with examples; otherwise, we shall restrict ourselves to a table of information needed for their proper use. We shall give more extensive treatment in the following sections to the functions for the formatted output of LINT objects, for which we shall make use of the properties of the *stream* classes contained in the C++ standard library. Possible applications, in particular for formatted output of objects of user-defined classes, are given rather short shrift in many C++ textbooks, and we are going to take the opportunity to explicate the construction of the functions needed to output our LINT objects.

15.1 Arithmetic

The following member functions implement the fundamental arithmetic operations as well as modular operations for calculation in residue class rings over the integers as accumulator operations: The object to which a called function belongs contains the function result as implicit argument after its termination. Accumulator functions are efficient, since they operate to the greatest extent

without internal auxiliary objects and thus save unnecessary assignments and calls to constructors.

For the cases in which a free assignment of the results of calculations is unavoidable, or in which the automatic overwriting of the implicit argument of the member functions with the result is not desired, the member functions were extended by means of like-named analogous friend functions together with additional friend functions. These are not discussed further here, but are recorded in Appendix B. The treatment of possible error situations in `LINT` functions that can arise from the use of `CLINT` functions will be discussed in full in Chapter 16.

Before we list the public member functions, we consider first as an example of their implementation the functions

```
LINT& LINT::mexp (const LINT& e, const LINT& m );
```

and

```
LINT& LINT::mexp (USHORT e, const LINT& m);
```

for exponentiation, an operation for which C++, alas, offers no operator. The functions `mexp()` were constructed in such a way that the functions used are, according to the type of the operands, the C functions `mexpk_l()`, `mexpkm_l()`, `umexp_l()`, and `umexpm_l()`, optimized for this purpose (with the corresponding arithmetic friend functions we are likewise dealing with the exponentiation functions `wmexp_l()` and `wmexpm_l()` with `USHORT` base).

Function:	Modular exponentiation with automatic use of Montgomery exponentiation if the modulus is odd.
Syntax:	`const LINT&` `LINT::mexp (const LINT& e, const LINT& m);`
Input:	implicit argument (base) `e` (exponent) `m` (modulus)
Return:	pointer to the remainder
Example:	`a.mexp (e, m);`

```
const LINT& LINT::mexp (const LINT& e, const LINT& m)
{
  int error;
  if (status == E_LINT_INV) panic (E_LINT_VAL, "mexp", 0, __LINE__);
```