CHAPTER 8

Input, Output, Assignment, Conversion

> The numerals were now being converted automatically from base 2 to base 10 . . . 881, 883, 887, 907 . . . each one confirmed as a prime number.
>
> —Carl Sagan, *Contact*

WE BEGIN THIS CHAPTER WITH assignment, the simplest and also the most important function. To be able to assign to a `CLINT` object `a_l` the value of another `CLINT` object `b_l`, we require a function that copies the digits of `b_l` to the reserved storage space for `a_l`, an event that we shall call *elementwise assignment*. It will not suffice merely to copy the address of the object `b_l` into the variable `a_l`, since then both objects would refer to the same location in memory, namely that of `b_l`, and any change in `a_l` would be reflected in a change in the object `b_l`, and conversely. Furthermore, access to the area of memory addressed by `a_l` could become lost.

We shall return to the problems of elementwise assignment in the second part of this book when we concern ourselves with the implementation of the assignment operator "=" in C++ (see Section 14.3).

The assignment of the value of a `CLINT` object to another `CLINT` is effected with the function `cpy_l()`.

Function:	copy a `CLINT` object as an assignment
Syntax:	`void cpy_l (CLINT dest_l, CLINT src_l);`
Input:	`src_l` (assigned value)
Output:	`dest_l` (destination object)

```
void
cpy_l (CLINT dest_l, CLINT src_l)
{
 clint *lastsrc_l = MSDPTR_L (src_l);
 *dest_l = *src_l;
```

In the next step leading zeros are found and then ignored. At the same time, the number of digits of the target object is adjusted.

```
 while ((*lastsrc_l == 0) && (*dest_l > 0))
  {
   --lastsrc_l;
   --*dest_l;
  }
```

Now the relevant digits of the source object are copied into the goal object. Then the function is terminated.

```
 while (src_l < lastsrc_l)
  {
   *++dest_l = *++src_l;
  }
}
```

The exchange of the values of two `CLINT` objects can be accomplished with the help of the macro `SWAP_L`, the FLINT/C variant of the macro `SWAP`, which manages in an interesting way to accomplish the exchange of two variables using `XOR` operations without the requirement of intermediate storage in a temporary variable:

```
#define SWAP(a, b) ((a)^=(b), (b)^=(a), (a)^=(b))
#define SWAP_L(a_l, b_l) \
  (xor_l((a_l), (b_l), (a_l)), \
   xor_l((b_l), (a_l), (b_l)), \
   xor_l((a_l), (b_l), (a_l)))
```