CHAPTER 16

Error Handling

O hateful error, melancholy's child!
—Shakespeare, *Julius Caesar*

16.1 (Don't) Panic ...

The C++ functions presented in the foregoing chapters embody mechanisms for analyzing whether during the execution of a called C function an error or other situation has occurred that requires a particular response or at least a warning. The functions test whether the passed variables have been initialized and evaluate the return value of the called C functions:

```
LINT f (LINT arg1, LINT arg2)
{
  LINT result;
  int err;

  if (arg1.status == E_LINT_INV)
    LINT::panic (E_LINT_VAL, "f", 1, __LINE__);
  if (arg2.status == E_LINT_INV)
    LINT::panic (E_LINT_VAL, "f", 2, __LINE__);
  // Call C function to execute operation; error code is stored in err
  err = f_l (arg1.n_l, arg2.n_l, result.n_l);
  switch (err)
    {
      case 0:
        result.status = E_LINT_OK;
        break;
      case E_CLINT_OFL:
        result.status = E_LINT_OFL;
        break;
      case E_CLINT_UFL:
        result.status = E_LINT_UFL;
        break;
      default:
        LINT::panic (E_LINT_ERR, "f", err, __LINE__);
    }
  return result;
}
```

If the variable status contains the value E_LINT_OK, then this is the optimal case. In less happy situations, in which overflow or underflow has occurred in a C function, the variable status is set to the appropriate value E_LINT_OFL or E_LINT_UFL. Since our C functions already react to an overflow or underflow with a reduction modulo $N_{\max} + 1$ (cf. page 20), in such cases the functions terminate normally. The value of the variable status can then be queried with the member function

```
LINT_ERRORS LINT::Get_Warning_Status (void);
```

Furthermore, we have seen that the LINT functions always call a function with the well-chosen name panic() when the situation gets too hot to handle. The task of this member function is first of all to output error messages, so that the user of the program is made aware that something has gone awry, and secondly to ensure a controlled termination of the program. The LINT error messages are output via the stream cerr, and they contain information about the nature of the error that has occurred, about the function that has detected the error, and about the arguments that have triggered the error. In order that panic() be able to output all of this information, such information arriving from the calling function must be delivered, as in the following example:

```
LINT::panic (E_LINT_DBZ, "%", 2, __LINE__);
```

Here it is announced that a division by zero in the operator "%" has appeared in the line specified by the ANSI macro __LINE__, caused by the operator's argument number 2. The arguments are indicated as follows: 0 always denotes the implicit argument of a member function, and all other arguments are numbered from left to right, beginning with 1. The LINT error routine panic() outputs error messages of the following type:

Example 1: Use of an uninitialized LINT object as argument.

```
critical run-time error detected by class LINT:
Argument 0 in Operator *= uninitialized, line 1997
ABNORMAL TERMINATION
```

Example 2: Division by a LINT object with the value 0.

```
critical run-time error detected by class LINT:
Division by zero, operator/function/, line 2000
ABNORMAL TERMINATION
```

The functions and operators of the LINT class recognize the situations listed in Table 16-1.