Essential C++ Errata -- 1st Printing

The following is a listing by chapter of the *Essential C++* errata that I am aware of. If you come across any corrections that are not listed, please drop me a line at *slippman@objectwrite.com*. Thanks! stan

0.1 Preface

Page ix:
*to read the raw Disney camera information for a scene and generate a camera node that could be plugged into **[not in to]** the Houdini animation package. I wrote it in C++, of course. It worked.*

Page xii: Acknowledgments
Clovis Tondo has been a reviewing fixture for my texts since the very first edition of the C++ Primer. His reviews have always proved extremely helpful, and it is with some embarrassment that I discover that I accidentally left his name off the Acknowledgments.

0.2 Chapter 1: Basic C++ Programming

Page 10 has one typo and one elided word. Bummer. The segment currently reads as follows, with the corrected and added word in bold:

*Alternatively, we can concatentate **[read: concatenate]** single characters into a string:*

// ...

All the data objects defined so far **[read: are]** *modified during the course of our program. `go_for_it`, for example, eventually gets set to `false`. `usr_score` is potentially updated with each user guess.*

Page 19, there is a superfluous 'I' preceding After in the 3rd paragraph of the page:

IAfter **[read: After]** *a `case` label is matched, all the `case` labels following the matched `case` label are also executed unless we explicitly break off execution. This is what the `break` statement does. Why, you're probably asking, is the `switch` statement designed this way? Here is an example of this fall-through behavior being just right:*

Page 20, the code-fragement while-loop, hear the bottom of the page:

```
cout << "Want to try another sequence? (Y/N) "
char try_again;
cin >> try-again; [ read: try_again; ]
```

Page 21, the last code fragment, within the comment:

```
// reach here only if the word is
// greater than or equal min-size [ read: min_size ] ...
process_text( word );
```

0.3 Chapter 2: Procedural Programming

Page 44, 2nd line of the output example at top of page -- missing space:

```
vector before sort: 8 34 3 13 1 21 5 2
about to call swap! ix: 0 jx: 2swapping: [ read: 2 swapping ] 8 with 3
```

Page 46, oops, wrong object was on the right-hand side of the assignment:

```
int jval = 4096;
rval = ival [ read: jval ];
```

Page 58, oops, wrong prototype is listed:

```
void display_message( const string&, const vector<string>&,
                      ostream& = cout );
```

should read:

```
void display_message( const string&, const vector<double>& );
```

Page 62, Refuse from an edit -- superfluous ‘e’:

An enumerated type is defined by the keyword `enum` *followed by an optional identifier, such as* `ns_type`*. eThe* ***[should read: The]*** *items in the comma-separated list of named values within*

0.4 Chapter 3: Generic Programming

Page 85, misspelling of return object:

```
int count_occurs( const vector<int> &vec, int val )
{
   // ...
   return ocurs_count; [ should read: occurs_count ]
}
```

Page 86, misspelling:

Let’s display the Fibonacci series in a series of increasingly inpenetrable ***[should read: impenetrable]*** *disguises: each element added to itself, each element multipled by itself, each element added to*

Page 89, left off call operator on parameter:

```
    vector<int>::iterator iter =
          find_if( local_vec.begin(),
                   local_vec.end(),
                   bind2nd( greater<int>, [ should read: greater<int>(), ] val ));
```

Page 97, superfluous ‘I’ at end of statement:

```
   ostream_iterator<string> os( out_file, " " );
   copy( text.begin(), text.end(), os );I
}
```

0.5 Chapter 4: Object-Based Programming

Page 115, the constructor is generating one additional element:

```
Triangular::Triangular( int len, int beg_pos )
    : _length( len > 0 ? len : 1 ),
      _beg_pos( beg_pos > 0 ? beg_pos : 1 )
{
   _next = _beg_pos-1;
   // [ should read: + _length - 1 ]
   int elem_cnt = _beg_pos + _length - 1;

   if ( _elems.size() < elem_cnt )
        gen_elements( elem_cnt );
}
```

Page 117, the `gen_elems_to_value()` function is duplicating the last element -- correction is to move the increment of the index. Here is the corrected code:

```
void Triangular::
gen_elems_to_value( int value )
{
   int ix = _elems.size();

   if ( !ix ){
      _elems.push_back( 1 );
      ix = 1;
   }

   while ( _elems[ ix-1 ] < value && ix < _max_elems )
   {
      ++ix; [ this gets done before calculation ]
      _elems.push_back( ix*(ix+1)/2 );

   }

   if ( ix == _max_elems )
       cerr << "Triangular Sequence: oops: value too large "
             << value << " --  exceeds max size of "
             << _max_elems << endl;

}
```

Page 118, the `gen_elements()` function is duplicating the last element:

```
void Triangular::
gen_elements( int length )
{
   // ...
   if ( _elems.size() < length )
   {
      int ix = _elems.size() ? _elems.size()+1 : 1;

      // [ not ix <= length-1 ]
      for ( ; ix <= length; ++ix )
            _elems.push_back( ix*(ix+1)/2 );
      }
}
```

Page 121, the `check_integrity()` function is off by one -- note that the two versions on page 124 illustrating friendship also need to be revised -- note: this also occurs on pages 191-192 of Chapter 7 when we look at the throw clause

```
inline void Triangular_iterator::
check_integrity() const
{
    // [ not > ]
    if ( _index >= Triangular::_max_elems )
         throw iterator_overflow();

    // [ not > or _index ]
    if ( _index >= Triangular::_elems.size() )
         Triangular::gen_elements( _index+1 );
}
```

Page 125, the output listed at the end of Section 2.7 is, because of the above, off by one -- should read as follows:

When compiled and executed, this program generates the following output:

```
Triangular Series of 20 elements
1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210
```

Page 132, typo: should read lucas, not lucus:

```
num_sequence::PtrType
num_sequence::func_tbl[ num_seq ] =
{  0,
       &num_sequence::fibonacci,
       &num_sequence::pell,
       &num_sequence::lucas, // [ not lucus ]
       &num_sequence::triangular,
       &num_sequence::square,
       &num_sequence::pentagonal
};
```

Page 133, another off by one:

```
int num_sequence::elem( int pos )
{
   if ( ! check_integrity( pos ))
        return 0;

   if ( pos > _elem->size() )
      ( this->*_pmf )( pos ); // [ not pos-1 ]

   return (*_elem)[ pos-1 ];
}
```

0.6 Chapter 5: Object-Oriented Programming

Page 137, top, missing a close parentheses of an if-statement condition:

```
    if ( mat.is_late() )
         mat.assess_fine();

    // was: if ( mat.waiting_list();
    if ( mat.waiting_list() )
         mat.notify_available();
}
```

Page 141, ‘a’ of output should read ‘an’:

```
AudioBook::print() -- I am an [ not: a ] AudioBook object!
```

Page 146, superfluous ‘I’:

*IA **[read: A]** third access level, `protected`, identifies operations that are available to the inheriting classes but not to the general program. `check_integrity()` and `gen_elems()`,*

Page 148, two more superfluous ‘I’’s:

*Although this completes the definition of the abstract `num_sequence` base class, the class itself is incomplete. It provides an interface for the subsequently derived classes. IEach **[read: Each]** derived class provides the implementation that completes the `num_sequence` base class definition.*

*IThe **[read: The]** derived class consists of two parts: the subobject of its base class (consisting of the nonstatic base class data members, if any) and the derived class portion (consist*

Page 151, code is incorrect -- note, this is repeated in the template implementation of p. 184:

```
// incorrect: if ( _elems.size() < pos )
if ( _elems.size() <= pos )
   {
     int ix  = _elems.size();
     int n_2 = _elems[ ix-2 ];
     int n_1 = _elems[ ix-1 ];

     // incorrect: for ( ; ix < pos; ++ix )
     for ( ; ix <= pos; ++ix )
      {
           int elem = n_2 + n_1;
           _elems.push_back( elem );
           n_2 = n_1; n_1 = elem;
      }
   }
```

Page 153, oops -- wrong object:

```
bool num_sequence::
check_integrity( int pos, int size )
{
   // wrong: if ( pos <= 0 || pos > max_seq ){
   if ( pos <= 0 || pos > _max_elems ){
        // same as before ...
   }

   if ( pos > size )
        // gen_elems() is invoked through virtual mechanism
        gen_elems( pos );

   return true;
}
```

Page 154, another superfluous ‘I’:

IIt ***[read: It]*** *is always a good idea to test an implementation incrementally rather than wait until the entire code base is complete to see whether the darn thing works. Not only*

Page 161, typo:

be a `const` *member function. The derived class instance is a non-*`const` *member function. Is this discepancy* ***[read: discrepancy]*** *significant? Unfortunately, it is. Here’s a simple illustration:*

Page 163, bottom line: superfluous ‘T’:

only the base class LibMat portion of `iWish` *can be copied into the memory reserved for* `object`*; Tthe* ***[read: the]*** *Book and AudioBook subobjects are sliced off.* `pointer` *and* `reference`

0.7 Chapter 6: Programming with Templates

Page 169, typo:

```
class string_BTnode {
public:
    //  ...
private:
   string _val;
   int _cnt;
   string_BTnode *_lchild; // not: int_BTnode *_lchild;
   string_BTnode *_rchild; // not: int_BTnode *_rchild;
};
```

Page 181, typos:

```
// not: class Fibonacci : public NumericSeries<len> {
class Fibonacci : public num_sequence<len> {
public:
```

instances of both the Fibonacci derived class and the num_sequence base class are generated with len bound to 16. Alternatively, we might parametermize ***[read: parameterize]*** *both the*

Page 184, incorrect code -- see Page 151 correction!
Page 185: off by one:

```
_len = len > 0 ? len : 1;
_beg_pos = beg_pos > 0 ? beg_pos : 1;

// not: pf( beg_pos+len, _elems );
pf( beg_pos+len-1, _elems );
```

0.8 Chapter 7: Exception Handling

Page 191-192, 195: incorrect code for the `check_integrity()` function-- see Page 121 for the correction!

0.9 Appendix A: Exercise Solutions

Page 207: typo:
than adequate. Second, we use the the ***[read: the]*** *standard library `strlen()` function to discover the size of `user_name`. The `cstring` header file holds the declaration of `strlen()`. If the*

Page 208 and 209: oops: old-style C++:

```
// oops: for ( int sum = 0, ix = 0; ix < icnt; ++ix )
int sum = 0;
for ( int ix = 0; ix < icnt; ++ix )
      sum += ia[ ix ];

int average = sum / icnt;
```

Page 210: typo:

```
ofstream out_file("C:\\My Documents\\text.sort" );
if ( ! out_file )
   // not: { cerr << "oops! unable to open input file\n"; return -2; }
   { cerr << "oops! unable to open output file\n"; return -2; }
```

Page 247: declaration of template friend function reflects the syntax of ‘classic’ C++ -- supported by Visual C++ 6.0 and the 7.3 SGI C++ compiler. The Standard C++ syntax looks as follows:

```
template <typename elemType> class Matrix;

// forward declaration required
template <typename elemType>
Matrix< elemType >
operator+( const Matrix<elemType> &m1, const Matrix<elemType> &m2 );

template <typename elemType>
Matrix<elemType>
operator*( const Matrix<elemType> &m1, const Matrix<elemType> &m2 );
```

```
template <typename elemType>
class Matrix
{
       // NOTE THE DIFFERENCE HERE ...
       friend Matrix<elemType>
       operator+<elemType>( const Matrix<elemType>&,
                            const Matrix<elemType>& );

       friend Matrix< elemType >
       operator*<elemType>( const Matrix<elemType>&,
                            const Matrix<elemType>& );
```

0.10 Appendix B: Generic Algorithms Handbook