ERRATA

to the book

Introduction to Programming with C++ for Engineers

by Bogusław Cyganek

Wiley 2021

First of all, I would like to thank all the people who were so kind and sent me information about the errors and imperfections they noticed in the text.

Writing a book, including all stages of its production, is a very large and long task, for which I take full responsibility. Therefore, I apologize for any imperfections, ambiguities or errors that have crept into the text. I will try to correct them and will post corrections, both here and in the code on *github*, if necessary.

Table 1. Found errors and their corrections.

Page, lines, what	It should be

Pg. 32, code line no. 26
 	
code line no. 49
	`if( r <= 0.0 || r >= 100.0 )  // operator || means the logical OR`
	
	`if( m <= 0.0 || m > 100 )`

Pg. 35, code line no. 22 Pg. 38, code line no. 22	`if( std::isalpha( c ) )  // isalpha( c ) returns true if c is alphabetic`

Pg. 50, third column in the table “Initialization by assignment …”	Let's notice that first defining a variable, then doing assignment in the second step is neither save, nor efficient.
Pg. 66, text line no. 11 from the top	Then, on lines [32–35], the sum of only the non-diagonal elements of m is computed.
Pg. 69, 3rd column, line no. 7 from the top	Then, all of the text is joined together by `accumulate`:
Pg. 98, third column in the table, last line with code	`cout << "Error - Wrong argument" << endl;`
Pg. 35, code line no. 11	`int main()`
Pg. 125, code line no. 3	`int FiboRecursive( const int n )`
Pg. 125, 1-2 lines in the last paragraph	The function does not change anything, so its input parameter is const int
Pg. 128, code line no. 11	`[] ( const double cm ) { return 0.3937 * cm; };`
Pg. 129, bullet line no. 1 Figure caption	Capture – Introduces external objects into the scope of the lambda Lambda expressions are composed of three parts: capture, formal arguments list, and optional return type. The lambda body is contained within braces {}.
Pg. 131, last line of text	capture to bring `cert` into its scope:

Pg. 132, code line no. 66	`// [ cert ] capture to access a copy of cert`
Pg. 132, 2nd line of text from the bottom	When using a lambda, it is very important is to properly specify the type of its capture.
Pg. 133-137, 1st pane in the 1st column	Capture
Pg. 134, code lines no. 20-21	`return static_cast< double >( std::rand() ) * max_val    // static_cast to make` `              / ( static_cast< double >( rm ) + 1. );    // division on double`
Pg. 138, the first row in the table	Lambda capture variants
Pg. 140, 4th line of text from the bottom	There are two more functions. The first, defined on line [10], simply computes the square of
Pg. 150, code line no. 34	`for( Dim ac = 0; ac < a_cols; ++ ac )`
Pg. 160, the last line of text	Which computes the roots of the quadratic equation. Both are implemented outside of the class definition.
Pg. 162, the 3rd line of text from the top	However, this does not mean that this class is fully implemented and finished.

Pg. 196, the first bullet lines	Let's introduce an *l-value* and an *r-value*: ▪ *l-value* – Informally, an object that has a name and can be taken address of, so it can be used on the left side of the assignment operator (=), e.g. x in `const int x = 10;` if `const` is omitted, then x becomes a modifiable *l-value*; an *l-value* can be converted to an *r-value*, e.g. if placed on the right side of = ▪ *r-value* – Not an *l-value*, such as no-name, temporary, literal constant, with no specific memory location objects; cannot be on the left side of the assignment, e.g. `10` in `const int x = 10;` cannot be converted to an *l-value* (https://en.cppreference.com/w/cpp/language/value_category)
Pg. 200, 3rd row, 2nd line of code	`// [=] capture denotes accessing external objects by value` Lambda functions are used to define local functions. The capture `[=]` allows the previous lambda to access
Pg. 214, code line no. 13-14 from the top	`void SetElem( int c, int r, T v ) { * (this->*data_offset)( c, r ) = v; }` `T GetElem( int c, int r ) { return * ((*this).*data_offset)( c, r ); }`
Pg. 225, code line no. 7 from the top	`const unsigned short kPoly = 0xE0;`
Pg. 225, text line no. 27 from the top	Moreover, when we declare (write) any of the special member functions, such as a destructor, copy constructor, or assignment operator, then probably we need to deliver all three of them at once - this is the so-called rule of three (or five, after adding a move constructor and a move assignment, which we'll talk about later).
Pg. 249, text line no. 17 from the top	strategy – see Section 4.6).
Pg. 254, text line no. 32 from the top	Declaring any special member function except a constructor,
Pg. 254, text line no. 39 from the top	rule of five (Section 4.4)

Pg. 267, text line no. 20 from the top text line no. 22 from the top	This time, the code – see lines [1–9] whereas the declaration goes in the header file
Pg. 283, code line no. 26	`// ~TinyCube() {}`
Pg. 285, code line no. 27	`// ~TinyCube() {}`
Pg. 326, the first bullet lines	■ Big endianness – Most significant byte at the lowest address ■ Little endianness – Most significant byte at the highest address
Pg. 328, text line no. 3 from the top	This can be achieved by providing the `: 4` bit specifier, which defines
Pg. 346, text line no. 20 from the top	If providing a custom copy constructor, an assignment operator or destructor, also write all the other special functions.
Pg. 351, 2nd column, text line no. 21 from the top	C++ program. Non-local statics are
Pg. 412, 2nd column, ‘depth’ entry	Returns the number of directories from the starting to the currently iterated one
Pg. 427, code line no. 120 Pg. 428, code line no. 139	`// connect through the class member variable`
Pg. 440, code line no. 17	`range( const T from, const T end, const T step = 1 )`

Pg. 444, text line no. 15 from the top	These are necessary to obtain a common language definition that allow us to implement
Pg. 452, text line no. 4 from the top	reflects the rules of operator precedence and associativity including parentheses.
Pg. 452, text line no. 27 from the top	expressions, fulfilling the precedence and associativity rules of operators without using parentheses.
Pg. 452, text line no. 27 from the top	Recall that in the integer representation, the LSB value was 1, so all intermediate values could be precisely represented. But in the case of a fraction, the LSB value
Pg. 521, text line no. 4 from the bottom	example, the associative law may not hold
Pg. 527, text line no. 5 from the top	condition in Eq. (7.30) is fulfilled for a certain value of the threshold
Pg. 544, text line no. 3 from the top text line no. 20 from the top	starting at index n = 0, provided in its capture. Only a lambda function declared mutable can change values passed in its capture
Pg. 554, text line no. 10 from the bottom	Thread 1 is faster and manages to execute lines [3–6] of its code.
Pg. 564, text line no. 8 from the bottom	sincetithisisthetotalsumofalloftheelementsthatareaddedintheloop.
Pg. 571, code line no. 3 from the top	`// its lowest local value (reduction for min)`
Pg. 572, first row in Table 8.3	$MSE = \frac{1}{N}\sum_{i=0}^{N-1}\left(u[i]-v[i]\right)^2$

Pg. 573, code line 10 from the bottom	`// Executes simultaneously with the next section`
Pg. 574, code line 1 from the top	`// Executes simultaneously with the previous section`