C++ Programming
from Beginner to Expert

Chapter 7: Class Templates array and vector and Catching Exceptions

Milad Molaee

AMIRKABIR
University of Technology

July 23, 2022

Outline

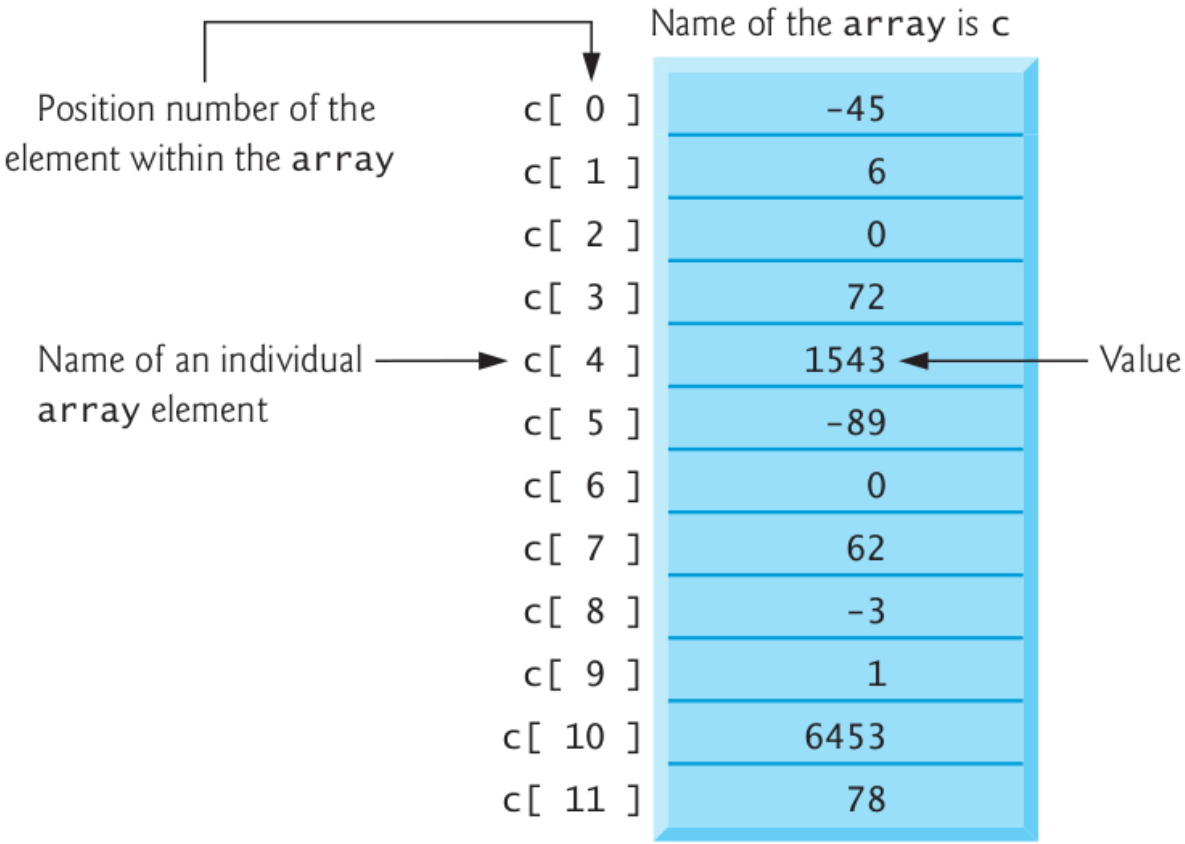
Introduction

This chapter introduces the topic of data structures—collections of related data items. We discuss arrays, which are fixed-size collections consisting of data items of the same type, and vectors, which are collections (also of data items of the same type) that can grow and shrink dynamically at execution time. Both array and vector are C++ standard library class templates. To use them, you must include the <array> and <vector> headers, respec- tively. After discussing how arrays are declared, created and initialized, we present examples that demonstrate several common array manipulations. We show how to search arrays to find particular elements and sort arrays to put their data in order. We build two versions of an instructor GradeBook case study that use arrays to main- tain sets of student grades in memory and analyze student grades. We introduce the excep- tion-handling mechanism and use it to allow a program to continue executing when it attempts to access an array or vector element that does not exist.

arrays

AMIRKABIR University of Technology

An array is a contiguous group of memory locations that all have the same type. To refer to a particular location or element in the array, we specify the name of the array and the position number of the particular element in the array. Figure 7.1 shows an integer array called c that contains 12 elements. You refer to any one of these elements by giving the array name followed by the particular element's posi- tion number in square brackets ([]). The position number is more formally called a sub- script or index (this number specifies the number of elements from the beginning of the array). The first element has subscript 0 (zero) and is sometimes called the zeroth ele- ment. Thus, the elements of array c are c[0] (pronounced "c sub zero"), c[1], c[2] and so on. The highest subscript in array c is 11, which is 1 less than the number of elements in the array (12). array names follow the same conventions as other variable names.



A subscript must be an integer or integer expression (using any integral type). If a pro- gram uses an expression as a subscript, then the program evaluates the expression to deter- mine the subscript. For example, if we assume that variable a is equal to 5 and that variable b is equal to 6, then the statement

```
c[a + b] += 2;
```

adds 2 to array element c[11]. A subscripted array name is an lvalue—it can be used on the left side of an assignment, just as non-array variable names can. Let's examine array c in Fig. 7.1 more closely. The name of the entire array is c. Each array knows its own size, which can be determined by calling its size member func- tion as in c.size(). Its 12 elements are referred to as c[0] to c[11]. The value of c[0] is -45, the value of c[7] is 62 and the value of c[11] is 78. To print the sum of the values contained in the first three elements of array c, we'd write

```
cout << c[0] + c[1] + c[2] << endl;
```

To divide the value of c[6] by 2 and assign the result to the variable x, we'd write

```
x = c[6] / 2;
```

Precedence and associativity of the operators introduced to this point

AMIRKABIR University of Technology

Operator	Associativity	Type
`::` `()`	left to right	primary
`()` `[]` `++` `--` `static_cast<`*type*`>(`*operand*`)`	left to right	postfix
`++` `--` `+` `-` `!`	right to left	prefix (unary)
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<<` `>>`	left to right	insertion/extraction
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`=` `+=` `-=` `*=` `/=` `%=`	right to left	assignment
`,`	left to right	comma

Declaring *arrays*

arrays occupy space in memory. To specify the type of the elements and the number of elements required by an array use a declaration of the form

```
array<type, arraySize> arrayName;
```

The notation <type, arraySize> indicates that array is a class template. The compiler reserves the appropriate amount of memory based on the type of the elements and the arraySize. (Recall that a declaration which reserves memory is more specifically known as a defini- tion.) The arraySize must be an unsigned integer. To tell the compiler to reserve 12 ele- ments for integer array c, use the declaration

```
array<int, 12> c; // c is an array of 12 int values
```

arrays can be declared to contain values of most data types. For example, an array of type string can be used to store character strings.

Declaring an array and Using a Loop to Initialize the array's Elements

AMIRKABIR University of Technology

The following examples demonstrate how to declare, initialize and manipulate arrays.

Declaring an array and Using a Loop to Initialize the array's Elements

The program in Fig. 7.3 declares five-element integer array n (line 9). Line 5 includes the <array> header, which contains the definition of class template array. Lines 12–14 use a for statement to initialize the array elements to zeros. Like other non-static local vari- ables, arrays are not implicitly initialized to zero (static arrays are). The first output statement (line 16) displays the column headings for the columns printed in the subse- quent for statement (lines 19–21), which prints the array in tabular format. Remember that setw specifies the field width in which only the next value is to be output.

```cpp
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;

int main()
{
    array<int, 5> n; // n is an array of 5 int values

    // initialize elements of array n to 0
    for (size_t i{0}; i < n.size(); ++i)
    {
        n[i] = 0; // set element at location i to 0
    }
    cout << "Element" << setw(10) << "Value" << endl;

    // output each array element's value
    for (size_t j{0}; j < n.size(); ++j)
    {
        cout << setw(7) << j << setw(10) << n[j] << endl;
    }
}
```

output

```
Element  Value
      0      0
      1      0
      2      0
      3      0
      4      0
```

Initializing an array in a Declaration with an Initializer List

AMIRKABIR University of Technology

```cpp
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;

int main()
{
    array<int, 5> n{32, 27, 64, 18, 95}; // list initializer
    cout << "Element" << setw(10) << "Value" << endl;

    // output each array element's value
    for (size_t i{0}; i < n.size(); ++i)
    {
        cout << setw(7) << i << setw(10) << n[i] << endl;
    }
}
```

output

```
Element  Value
      0     32
      1     27
      2     64
      3     18
      4     95
```

Specifying an array's Size with a Constant Variable

Example 7.3

```
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;
int main()
{
    // constant variable can be used to specify array size
    const size_t arraySize{5};    // must initialize in declaration

    array<int, arraySize> values; // array values has 5 elements

    for (size_t i{0}; i < values.size(); ++i)
    { // set the values
        values[i] = 2 + 2 * i;
    }

    cout << "Element" << setw(10) << "Value" << endl;

    // output contents of array s in tabular format
    for (size_t j{0}; j < values.size(); ++j)
    {
        cout << setw(7) << j << setw(10) << values[j] << endl;
    }
}
```

output

```
Element  Value
      0      2
      1      4
      2      6
      3      8
      4     10
```

Note

Line 8 uses the const qualifier to declare a constant variable `arraySize` with the value 5. Constant variables are also called *named constants* or *read-only* variables. A constant variable must be initialized when it's declared and cannot be modified thereafter. Attempting to modify `arraySize` after it's initialized, as in

```
8    const size_t arraySize{5};    // must initialize in declaration
```

results in the following errors from Visual C++, GNU C++ and LLVM, respectively:

- Visual C++: `'arraySize': you cannot assign to a variable that is const`
- GNU C++: `error: assignment of read-only variable ‘x’`
- LLVM: `Read-only variable is not assignable`

Summing the Elements of an array

AMIRKABIR University of Technology

Often, the elements of an array represent a series of values to be used in a calculation. For example, if the elements of an array represent exam grades, a professor may wish to total the elements of the array and use that sum to calculate the class average for the exam. The program in Fig. 7.6 sums the values contained in the four-element integer array a. The program declares, creates and initializes the array in line 9. The for statement (lines 13–15) performs the calculations. The values being supplied as initializers for array a also could be read into the program—for example, from the user at the keyboard or from a file on disk (see Chapter 14, File Processing). For example, the for statement

```
for (size_t j{0}; j < a.size(); ++j)
{
    cin >> a[j];
}
```

reads one value at a time from the keyboard and stores the value in element a[j].

```cpp
#include <iostream>
#include <array>
using namespace std;

int main()
{
    const size_t arraySize{4}; // specifies size of array
    array<int, arraySize> a{10, 20, 30, 40};
    int total{0};

    // sum contents of array a
    for (size_t i{0}; i < a.size(); ++i)
    {
        total += a[i];
    }

    cout << "Total of array elements: " << total << endl;
}
```

Output

```
Total of array elements: 100
```

Using a Bar Chart to Display array Data Graphically


```cpp
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;
int main()
{
    const size_t arraySize{11};
    array<unsigned int, arraySize> n{0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1};
    cout << "Grade distribution:" << endl;
    // for each element of array n, output a bar of the chart
    for (size_t i{0}; i < n.size(); ++i)
    {
        // output bar labels ("0-9:", ..., "90-99:", "100:")
        if (0 == i) {
            cout << " 0-9: ";
        } else if (10 == i) {
            cout << " 100: ";
        } else {
            cout << i * 10 << "-" << (i * 10) + 9 << ": ";
        }
        // print bar of asterisks
        for (unsigned int stars{0}; stars < n[i]; ++stars)
        {
            cout << '*';
        }
        cout << endl; // start a new line of output
    }
}
```

Output

```
Grade distribution:
  0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Using the Elements of an array as Counters

Sometimes, programs use counter variables to summarize data, such as the results of a sur- vey. In Fig. 6.7, we used separate counters in our die-rolling program to track the number of occurrences of each side of a die as the program rolled the die 60,000,000 times. An ar- ray version of this program is shown in Fig. 7.8. This version also uses the new C++11 random-number generation capabilities that were introduced in Section 6.9. Figure 7.8 uses the array frequency (line 17) to count the occurrences of die value. The single statement in line 21 of this program replaces the entire switch statement in lines 22–43 of Fig. 6.7. Line 21 uses a random value to determine which frequency element to increment during each iteration of the loop. The calculation in line 21 produces a random subscript from 1 to 6, so array frequency must be large enough to store six counters. We use a seven-element array in which we ignore frequency[0]—it's clearer to have the die face value 1 increment frequency[1] than frequency[0]. Thus, each face value is used directly as a subscript for array frequency. We also replace lines 46–51 of Fig. 6.7 by looping through array frequency to output the results (Fig. 7.8, lines 27–29).

```cpp
#include <iostream>
#include <iomanip>
#include <array>
#include <random>
#include <ctime>
using namespace std;

int main()
{
    // use the default random-number generation engine to
    // produce uniformly distributed pseudorandom int values from 1 to 6
    default_random_engine engine(static_cast<unsigned int>(time(0)));
    uniform_int_distribution<unsigned int> randomInt(1, 6);

    const size_t arraySize{7};                    // ignore element zero
    array<unsigned int, arraySize> frequency{}; // initialize to 0s

    // roll die 60,000,000 times; use die value as frequency index
    for (unsigned int roll{1}; roll <= 60000000; ++roll)
    {
        ++frequency[randomInt(engine)];
    }
    cout << "Face" << setw(13) << "Frequency" << endl;

    // output each array element's value
    for (size_t face{1}; face < frequency.size(); ++face)
    {
        cout << setw(4) << face << setw(13) << frequency[face] << endl;
    }
}
```

Output

Face	Frequency
1	9997901
2	9999110
3	10001172
4	10003619
5	9997606
6	10000592

increment during each iteration of the loop. The calculation in line 21 produces a random subscript from 1 to 6, so array frequency must be large enough to store six counters. We use a seven-element array in which we ignore frequency[0]—it's clearer to have the die face value 1 increment frequency[1] than frequency[0]. Thus, each face value is used directly as a subscript for array frequency. We also replace lines 46–51 of Fig. 6.7 by looping through array frequency to output the results (Fig. 7.8, lines 27–29).

Using arrays to Summarize Survey Results

AMIRKABIR University of Technology

Our next example uses arrays to summarize the results of data collected in a survey. Con- sider the following problem statement:

Example

Twenty students were asked to rate on a scale of 1 to 5 the quality of the food in the student cafeteria, with 1 being "awful" and 5 being "excellent." Place the 20 responses in an integer array and determine the frequency of each rating.

This is a popular type of array-processing application (Fig. 7.9). We wish to summarize the number of responses of each rating (that is, 1–5). The array responses (lines 14–15) is a 20-element integer array of the students' responses to the survey. The array respons- es is declared const, as its values do not (and should not) change. We use a six-element array frequency (line 18) to count the number of occurrences of each response. Each el- ement of the array is used as a counter for one of the survey responses and is initialized to zero. As in Fig. 7.8, we ignore frequency[0].

```cpp
#include <iostream>
#include <iomanip>
#include <array>
using namespace std;

int main() {
    // define array sizes
    const size_t responseSize{20}; // size of array responses
    const size_t frequencySize{6}; // size of array frequency

    // place survey responses in array responses
    const array<unsigned int, responseSize> responses{
        1, 2, 5, 4, 3, 5, 2, 1, 3, 1, 4, 3, 3, 3, 2, 3, 3, 2, 2, 5};

    // initialize frequency counters to 0
    array<unsigned int, frequencySize> frequency{};

    // for each answer, select responses element and use that value
    // as frequency subscript to determine element to increment
    for (size_t answer{0}; answer < responses.size(); ++answer)
    {
        ++frequency[responses[answer]];
    }
    cout << "Rating" << setw(12) << "Frequency" << endl;

    // output each array element's value
    for (size_t rating{1}; rating < frequency.size(); ++rating)
    {
        cout << setw(6) << rating << setw(12) << frequency[rating] << endl;
    }
}
```

Output

```
Rating   Frequency
     1           3
     2           5
     3           7
     4           2
     5           3
```

Static Local arrays and Automatic Local arrays

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

```cpp
#include <iostream>
#include <array>
using namespace std;
void staticArrayInit();    // function prototype
void automaticArrayInit(); // function prototype
const size_t arraySize{3};

int main()
{
    cout << "First call to each function:\n";
    staticArrayInit();
    automaticArrayInit();

    cout << "\n\nSecond call to each function:\n";
    staticArrayInit();
    automaticArrayInit();
    cout << endl;
}

// function to demonstrate a static local array
void staticArrayInit(void)
{
    // initializes elements to 0 first time function is called
    static array<int, arraySize> array1; // static local array
    cout << "\nValues on entering staticArrayInit:\n";

    // output contents of array1
    for (size_t i{0}; i < array1.size(); ++i)
    {
        cout << "array1[" << i << "] = " << array1[i] << " ";
    }
```

```
32
33      cout << "\nValues on exiting staticArrayInit:\n";
34
35      // modify and output contents of array1
36      for (size_t j{0}; j < array1.size(); ++j)
37      {
38          cout << "array1[" << j << "] = " << (array1[j] += 5) << " ";
39      }
40  }
41
42  // function to demonstrate an automatic local array
43  void automaticArrayInit(void)
44  {
45      // initializes elements each time function is called
46      array<int, arraySize> array2{1, 2, 3}; // automatic local array
47      cout << "\n\nValues on entering automaticArrayInit:\n";
48
49      // output contents of array2
50      for (size_t i{0}; i < array2.size(); ++i)
51      {
52          cout << "array2[" << i << "] = " << array2[i] << " ";
53      }
54
55      cout << "\nValues on exiting automaticArrayInit:\n";
56
57      // modify and output contents of array2
58      for (size_t j{0}; j < array2.size(); ++j)
59      {
60          cout << "array2[" << j << "] = " << (array2[j] += 5) << " ";
61      }
62  }
```

Output

```
First call to each function:

Values on entering staticArrayInit:
array1[0] = 0 array1[1] = 0 array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10 array1[1] = 10 array1[2] = 10

Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8
```

Range-Based *for* Statement

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Sorting and Searching

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Multidimensional *arrays*

AMIRKABIR University of Technology

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Introduction to C++ Standard Library Class Templat

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Summery and Conclusion

AMIRKABIR University of Technology

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Exercises

AMIRKABIR University of Technology

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

First Program in C++: Printing a Line of Text

AMIRKABIR University of Technology

Consider a simple program that prints a line of text (Fig. 2.1). This program illustrates several important features of the C++ language. The text in lines 1–10 is the program's source code. The line numbers are not part of the source code.

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // enables program to output data to the screen
4
5  // function main begins program execution
6  int main() {
7  std::cout << "Welcome to C++!\n"; // display message
8
9  return 0; // indicate that program ended successfully
10 } // end function main
```

output

```
Welcome to C++!
```

The Stream Insertion Operator and Escape Sequences

AMIRKABIR University of Technology

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\'	Single quote. Used to print a single-quote character.
\"	Double quote. Used to print a double-quote character.

Arithmetic Operators

AMIRKABIR University of Technology

Operation	Arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	$\frac{x}{y}$ or x/y or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Precedence of Arithmetic Operators

AMIRKABIR University of Technology

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. For nested parentheses, such as in the expression `a * (b + c / (d + e))`, the expression in the innermost pair evalu ates first. [Caution: If you have an expression such as `(a + b) * (c - d)` in which two sets of parentheses are not nested, but appear "on the same level," the C++ Standard does not specify the order in which these parenthesized subexpressions will evaluate.]

`*`
`/`
`%`
	Multiplication
	Division
	Remainder
		Evaluated second. If there are several, they're evaluated left to right.

`+` `-`	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Decision Making: Equality and Relational Operators

AMIRKABIR University of Technology

Algebraic relational or equality operator	C++ relational or equality operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
$>$	`>`	`x > y`	x is greater than y
$<$	`<`	`x < y`	x is less than y
$\geq$	`>=`	`x >= y` or	x is greater than or equal to y
$\leq$	`<=`	`x <= y`	x is less than or equal to y
Equality operators			
$=$	`==`	`x == y`	x is equal to y
$\neq$	`!=`	`x != y`	x is not equal to y