The Rook's Guide to C++

Kickstarter Backer & Contributor Version

August 1, 2013

Contents

Dramatis Personæ

Managing Editor:

Jeremy A. Hansen, PhD, CISSP

Technical Editing & Typesetting:

Theodore M. Rolle, Jr.

Media & Outreach:

Matthew E. Russo

Cover Art & Graphic Design:

Allyson E. LeFebvre

Content Authors:

Tyler Atkinson, Troy M. Dundas, Connor J. Fortune, Jeremy A. Hansen, PhD, CISSP, Scott T. Heimann, Benjamin J. Jones, Michelle Kellerman, Michael E. Kirl, Zachary LeBlanc, Allyson E. LeFebvre, Gerard O. McEleney, Phung P. Pham, Megan Rioux, Alex Robinson, Kyle R. Robinson-O'Brien, Jesse A. Rodimon, Matthew E. Russo, Yosary Silvestre, Ryan S. Sutherland, James M. Verderico, Christian J. Vergnes, Rebecca Weaver, Richard Z. Wells, and Branden M. Wilson.

Funding & Support:

- Peter Stephenson, PhD, VSM, CISSP, CISM, FICAF, LPI at the Norwich University Center for Advanced Computing & Digital Forensics
- Andrew Pedley at Depot Square Pizza
- Theodore M. Rolle, Jr.

Kickstarter contributors:

Nathan Adams, Chris Aldrich, Jay Anderson, Kent Archie, Erik Arvedson, Astrolox, Phoebe Ayers, Papa Joe Barr, Julia Benson-Slaughter, Georgia Perimeter College, Patrick Berthon, Francis Bolduc, Greg Borenstein, Mark Braun, Patrick Breen, Igor Bronshteyn, Valdemar Bučilko, Ross Buckley, Nikita Burtsev, Jakob Bysewski, David Camara, Dave M. Campbell, Brian V. Campbell III, S. Canero, Serge Canizares, Andrew Carlberg, Casey B. Cessnun, Winston Chow, W. Jesse Clements, Greg Crawford, Sean Cristofori, Jordan G Cundiff, Michael David, Joseph Davies, Ashley Davis, David C. Dean, DJS, Carlton Doc Dodd, Phil Dodds, Dominic, Sankar Dorai, dryack, Matt DuHarte, Brandon Duong, Van Van Duong, Daniel Egger, Chris Fabian, Jorge F. Falcon, Tek Francis, Fuchsi, Steve Gannon, Michael Gaskins, Gavlig, Adam Gibson, Russell E. Gibson, Goldenwyrm, James Green, Brian J. Green, Casey Grooms, Vitalik Gruntkovskiy, Vegar Guldal, Felix Gutbrodt, Jeremy Gwinnup, Beau T. Hahn, Paul R. Harms - Norwich 1975, Corey H. Hart, MBA, Aaron A. Haviland, Josh Heffner, Greg Holland, Henry Howard, Mark V Howe, Ivaliy Ivanov, Matt Jadud, Joseph Jaoudi, Tim R. Johnson, Ibi-Wan Kentobi, Mark King, Mitchell Kogut III, Sigmund Kopperud, Michael Korolev, Jamie Kosoy, Aria Kraft, Alexander Týr Kristjánsson, Richard Kutscher, Eric Laberge, John Lagerquist, Philip Lawrence, Mark Brent Lee, John and Nancy LeFebvre, Nevin :-) Liber, Jonathan Lindquist, Thomas Lockney, Stuart A. MacGillivray, Dr. Pedro Maciel, Troels Holstein Madsen, William Marone, Fred Mastropasqua,

Miles Mawyer, michael mazzello, Ryan McDonough, Matthew McFadden, John McIntosh II, Sean McNamara, mdsaravanan, Brandon Meskimen, Andrew Mike, G.F. Miller IV, Marcus Millin, Salvador Torres Morales, Danny Morgan, Ken Moulton, Aaron Murray, mvi, Jon Nebenfuhr, Philip K. Nicholson, chris nielsen, Pontus Nilsson, Mike Noble, Aleksander R. Nordgarden-Rødner, Greg O'Hanlon, Doug Otto, Randy Padawer, Ph.D., J Palmer, Tasos Papastylianou, Paul, James Pearson-Kirk, Matthew Peterson, Grigory Petrov, pezmanlou, Joachim Pileborg, Kyle Pinches, pkh, Mary Purdey, Marshall Reeves, Matthew Ringman, Craig D. Robbins, Antonio Rodriguez, Armando Emanuel Roggio, Victor Suarez Rovere, Christian Sandberg, Jaymes Sattler, Paolo Scalia, Patrice Scheidt, Daniel Schmitt, Levi Schuck, Raman Sharma_Himachali, Michael Shashoua, Daniel Shiffman, Clay Shirky, sillygoatgirl, Kevin J. Slonka, Brian Smith, Hazel Smith & Rebecca Twigg, Andrey Soplevenko, Kasper Souren, Derek A. Spangler, Speckman, Kellan St.Louis, Nick Stefanou, Steve, Andrew Stewart, Jeremy Sturdivant, Cyrille Tabary, Adam 8T Tannir, M Taylor, Telecat Productions, Aron Temkin, Mitchell Tilbrook, Nathan Tolbert, Devin M. Tomlinson - Vermont Born, Todd Trimble, Michiel van Slobbe, James A. Velez, Marco Verdecchia, David Walter, Lothar Werzinger, Wayne West, Sean Whaley '05 & M'08, Mark Wheeler, Tommy Widenflycht, Dylan Widis, Tony Williamitis, Adam M. Williams, Stephen D. Williams, Dylan Wilson, Wesley Wiser, wizzy, Sam Wright, Janet Hong Yam, and Jy Yaworski.

Chapter 1

History

Chapter 2

Sample Program

Chapter 3

Variables

Chapter 4

Constants

Chapter 5

Assignments

Assignments are a way for a user or a programmer to assign a value to a variable. The way we assign a value to a variable in C++ is different from how we would might do it in math. In mathematics we are allowed to say that x = 3 or 3 = x, but in C++ the only acceptable way to assign the value of 3 to x is to type x = 3. The = in the expression x = 3 is known as an *assignment operator*. This allows the program to set a variable⌧s value depending on its type. Here are some examples on setting a value to different types of variables:

int x = 4;

char alpha = 'A';

string word = Älphaï;

float y = 3.14;

We are able to declare variables and assign a value to those variables immediately by using the assignment operator. When we assign literal values to variables of type char, the value must be surrounded by single quotes (example: 'A'). When we assign values of type string, the literal value must be surrounded by double quotes (example: Älphä). We do not have to initialize the values of the variables, however. We can set them with a value later in the code like this: int myVal; //some code myVal = 0;

In all of the lines of code in this section where a variable is set using the assignment operator, the "thing that is being given a value" is known as an *lvalue*, and the expression on the right that is being stored in the variable is known as the *rvalue*. Literals such as `'A'` or 3 can never be an lvalue. Aside from literals, the rvalue can consist of other variables as well, like this:

myVal = myVal2;

Even though `myVal2` is a variable, we are only passing the *value stored in the variable*, not the variable itself. For example if `myVal2` had a value of 6, `myVal` would then have a value of 6 with the above code.

We can also store an arithmetic expression in a variable like this:

myVal = 5 + 6; //assigns myVal a value of 11 because 5 plus (+) 6 is 11

But we can't write

5 + 6 = myVal; // ERROR!

since 5 + 6 doesn't refer to a place where we can store a value. We can also combine arithmetic expressions with other variables to be stored in a variable like this:

myVal2 = 6; myVal = 4 + myVal2;

In this case, the variable `myVal` would be given a value of 10 because the variable `myVal2` was initialized with a value of 6 before the assignment was executed. The value of `myVal2` remains unchanged. Make sure that the variable `myVal`, the variable `myVal2`, and the value 4 are the same type. For example, the following code will result in an error:

int myValue = 4; int yourVal; string myString = ẅord̈;

yourVal = myValue + myString; // Adding an int to a string is // probably not what you meant!

When we try to combine different variable types, the compiler will get very mad at us. Some exceptions to this rule are if we try to combine `floats`, `ints`, and `doubles`. These types have the ability to be combined because they are all numeric values. Both

`doubles` and `floats` can hold values with a decimal point such as -3.14, 0.003, or 5.167289 whereas an `int` can only hold round values such as 2, -18, or 100. Refer to Chapter **??** for more information on converting between data types.

Chapter 6

Input

Chapter 7

Output

]

Chapter 8

Arithmetic

One of the most important things provided by C++ is the ability to do math. Everything a computer sees is a number. To a computer, its ability to do math and manipulate numbers is as essential to it as breathing is to us. (My apologies to anything not living that may be reading this).

The operators (+, -, *, /) in C++ are slightly different from what you may be used to from your second-grade math class. Addition is still a plus sign (+) and subtraction is still a minus sign (-). On the other hand, multiplication becomes an asterisk (*) and division becomes a forward slash (/). Think of it as 'over' as in 5 over 9 is the same as the fraction $5/9$.

To do math in C++, you will either want a variable to store the answer, or output the answer to the user.

The following code directly outputs the answer to the user:

cout « 9 + 2;

This code shows how to use a variable to store the answer:

Notice: When you use a variable to store an answer, the variable must come first in the equation (before the equal sign) and must be the only thing on the left side of the equation.

Other things to note: When you use more complicated equa-

tions, you can use parentheses to help. C++ does use a familiar order of operations (Parentheses, Exponents, Multiply, Divide, Add, and Subtract, or PEMDAS), but without the exponent operation (this topic is covered in Chapter **??**, Advanced Arithmetic). However, unlike in normal arithmetic, parentheses do not imply multiplication. `(4)(3)` does not mean the same as `4 * 3`. `(4)(3)` results in a syntax error and does not compile. The compiler returns an error message like this: `'error: '4' cannot be used as a function.'`

In C++, there are several methods of shortening and simplifying the code you're creating. The first is the increment operator (`++`), which is the found in the name of the language C++. This operator increases the value of the variable it's applied to by 1. Conversely, the decrement (`--`) operator decreases the value by 1.

Keep in mind that order does matter with these increment and decrement operators. They can be used as either prefixes or suffixes, but where you put the operator results in different behavior. Starting with similarities, C++ and ++C both increase value of C by one. The difference lies in when another variable is being set to that incremented value, such as `B = C++`. B will be set to C before C is incremented. `B = ++C` will cause B to be set to C+1, in a similar way to `B = 1 + C`.

int A; A = 4; A++;

//A contains 5 int A; A = 9; A−;

//A contains 8 int A, B; B = 7; A = B++;

//A contains 7, B contains 8 int A, B; B = 7; A = ++B;

//A contains 8, B contains 8 int A, B; B = 3; A = B−;

//A contains 3, B contains 2 int A, B; B = 3; A = −B;

//A contains 2, B contains 2

Compound assignment operators can decrease the amount you type and can make your code more readable. These are the operators +=, −=, * =, and / =. What makes these operators special is that they use the value you want to change in the operation. For example, x

+= 2 is equivalent to x = x + 2.

Keep in mind the order that was used, as this becomes important with subtraction and division. The variable being changed is equivalent to the two leftmost variables in the longhand expression. So, let⌧s say we have X and Y, and want to set X equal to the value of Y divided by the value of X. This is impossible with this method, as X /= Y is equivalent to X = X / Y, and Y /= X is equivalent to Y = Y / X.

Expression Equivalent To A *= 3; A = A * 3; B -= 5; B = B - 5; C /= 10; C = C / 10;

Code:

#include <iostream>

using namespace std;

int main() int a = 5, b = 10, c = 15, d = 20;

cout « ä + b = ¨< a + b « endl; cout « đ - c = ¨< d - c « endl; cout « ä * b = ¨< a * b « endl; cout « đ / a = ¨< d / a « endl;

Output:

a + b = 15 d - c = 5 a * b = 50 d / a = 4

Review Questions 1. Write a statement declaring two integer variables a and b and initialize them to 6 and 3 respectively.

2. Without changing the last line, fix the following code so there will be an output of 12. int a = 4, b= 2;

a = a + 2 * b; cout « a;

3.What is the output of the following? int a=2, b=5, c=6;

a++; b = b*a; c = (c-a) + 3; cout « a « endl; cout « b « endl; cout « c « endl; 4. What is the output for the following code? int a, b, c; a = 2; b = 8; c = 1; c = b - b; c = a + a; c = b * 8; b = b + b; c = a + c; b = a + b; a = a * c; b = a - c; c = b + a; cout « a « endl; cout « b « endl; cout « c « endl;

Homework Exercises 1.What is the output of the following code?

int a=4, b=2, c, d;

a = b + 3; b++; c = (b + 4) * 2; c = c + 2; d = a + b - 3; a++; a = a + 2 - b; b = b * 2; cout « ä=¨< a « endl; cout « b̈=¨< b « endl; cout « c̈=¨<

c « endl; cout « d̈=c̈< d « endl; 2.What is the output of the following code?

int m=3, n=2, x, y;

x = m + 5; m−; y = (m+4) / 3; n = n + 2; m = m + n / 2; m++; x = x * 2 - 3; y = y * 2; n = n + y * 3 cout « m̈=c̈< m « endl; cout « n̈=c̈< n « endl; cout « ẍ=c̈< x « endl; cout « ÿ=c̈< y « endl;

Review Exercises Answers 1. int a = 6, b = 3;

2. int a = 4, b= 2;

a = (a + 2) * b; cout « a;

3. 3 15 5

4. 132 66 198

Homework Exercises Answers 1. a=5 b=6 c=16 d=5

2. m=5 n=16 x=13 y=4

Further Reading

1.http://www.cplusplus.com/doc/tutorial/operators/
2.http://www.sparknotes.com/cs/c-plus-plus-fundament
basiccommands/section1.rhtml

Chapter 9

Comments

Chapter 10

Data types, conversion

Chapter 11

Conditionals

Chapter 12

Strings

Chapter 13

Loops

Chapter 14

Functions, Scope, and Blocks

14.1 Blocks

Since we've covered `if` statements and loops, let's go into more detail about the code that's contained within them. When you need to contain multiple lines of code, we've shown how to use braces. These braces will create a new layer in the code, and the lines within would be grouped into what is known as a compound statement, sometimes called a block.

```
int x;
cin << x;

if(x < 5)
{
  int y;
  cin << y;
  x += y; //Declares Y, asks user to define Y, then
      sets Y to X + Y
}

if(x > 5)
```

```
{
  int z;
  cin << z;
  x -= z; //Declares z, asks user to define z, then
      sets x to x - z
}

cout >> x;
//outputs either x-z, x-y, or 5
```

Take a look at the example above. There are two blocks here: the one for if x is less than 5, and the one for if x is greater in 5. Notice the variables declared in each, y and z. When these are declared, they are only usable within the blocks that they were declared. When that block reaches its end, they are lost to the rest of the program. This is because the scope of the variables within the blocks is limited to those blocks.

Scope is a topic we'll cover in more depth in the next chapter. For now, think of it as the difference between local and federal government. Local governments can freely request to utilize federal resources, but federal governments have a harder time accessing local government resources, and may forget the local government even exists.

Chapter 15

Problem Solving

Chapter 16

Testing

Chapter 17

Preprocessor

Chapter 18

Advanced Arithmetic

Advanced arithmetic in C++ includes math that can't be used in code without the use of the <cmath> library. This is math that goes above and beyond the standard operations: addition (+), subtraction (-), multiplication (*), and division (/). As we have seen before, some simple arithmetic might look like:

```
int x;
x = 1;
x += 5;
return x;
```

The variable x is declared as an integer. The next line sets it to one. The += operator adds five to x, which makes x contain six.

Doing simple operations like these does not require any special libraries or unusual commands. Any compiler can look at a +, -, *, or / in a line of code and know exactly what the programmer is expecting to happen. Some math requires a little extra help, though. In this case, help is the <cmath> library.

<cmath> is a library that is needed for trigonometric, hyperbolic, exponential, logarithmic, rounding, and absolute value functions. The <cmath> library is designed to make your life simple

and to make complicated math easier in C++. Using the `<cmath>` library in code is as simple as including it at the top of your source code file with the rest of your libraries:

```
#include <iostream>
#include <cmath>
```

After the inclusion of the `<cmath>` library, you can use certain mathematical functions in your code such as `pow(x,y)`, which raises parameter `x` to the power of parameter `y`, and `sqrt(z)`, which returns the square root of parameter `z`. In your first few C++ programs, you will probably not use the more advanced mathematical functions included in the `<cmath>` library, but for a full list of the mathematical functions, refer to "Further Reading" at the end of this chapter.

18.1 Examples

18.1.1 pow()

pow is the function that is called when a value or variable needs to be raised to a certain power. Take a look at this code and we'll break it down line by line:

```
int x, y;
x = 4;
y = pow (x+1, 3) + 6;
```

First off, we are initializing two variables, `x` and `y`. After that, we take `x` and set it to 4. Now we get to a more interesting section of code. We are asking the compiler to raise the value of `x` plus 1 to the power of 3, add 6, and then place the result in `y`. To use the pow function, you must understand its syntax. Here is the breakdown:

```
pow (starting value, power being raised)
```

So for `pow (x+1, 3) + 6`, we are raising the starting value `x + 1` to the power of 3. Before the power of 3 is applied to `x+1`, 1 needs to be added to `x`. In this case, it is the simple operation of 4+1, which nets us 5. After we get 5, we raise it to the 3rd power to get a value of 125. After we reach the value of 125, we are finished with the `pow` function and return to using normal math when we add 6 to 125 and get a value of 131.

Undoubtedly there are more complicated uses of the `pow` function, such as multiple uses of `pow` in the same line of code. You might use multiple `pow` operations in code that calculates the length of one side of a triangle using the Pythagorean Theorem. Look at the following code and see if you can figure out what the output value would be:

```
int x, y, z;
x = 3;
y = x + 1;
z = pow (x, 2) + pow (y, 2);
cout << z;
```

If you got 25, then you have the right answer! A breakdown reveals that after initializing the variables `x` and `y` and setting their values (3 for `x` and `x+1` for `y`), we raise each value to the power of 2. For visual reference,

```
z = pow (3, 2) + pow (x+1, 2);
```

gets changed to

```
z = 9 + 16;
```

z's value is set to 25. The pow operation is simple to use, and can make operations used in a program simpler from both a computing and visual stand point.

18.1.2 `sqrt()`

Square root operations are called in the program with `sqrt`. Take a look at the example below to see how it is called in a program:

```
int a, b;
a = 25;
b = sqrt(a);
```

`sqrt` is simpler than pow in that it only requires one parameter. Since `sqrt` returns a `float` or `double`, you should usually assign the result to a `float` or `double` variable, but in this example, `sqrt` returns exactly 5, so it can be converted to an `int` without any issues.

There are cases where both `sqrt` and pow are used in the same formula, such as when calculating the distance between two points. When writing such code, it is very important to keep track of the parentheses and to use correct syntax. One such syntax mistake is made when programmers think that C++ syntax is the same as algebraic syntax. This is *not* the case in C++!

```
int x = (5)(pow (3, 3) );
```

is NOT correct syntax. When the compiler sees this, it doesn't view it as multiplication, but instead as (according to a professional), "function shenanigans." It is important to be explicit with symbols relating to math in C++. So instead of the inaccurate code above, use:

```
int x = 5 * (pow (3, 3) );
```

Now, as an example, we will use code to compute the distance between two points on a plane. Refer to the code and the diagrams if you do not understand or get lost.

```
int x1, x2, y1, y2;
float dist;
x1 = 4;
y1 = 4;
x2 = 6;
y2 = 10;

dist = sqrt  (pow (x2 - x1, 2) + pow (y2-y1, 2));
cout << dist;
```

So your final answer after the calculation is executed is roughly 6.342555. Without the help of the advanced arithmetic operations, getting to this result would be a difficult, long, drawn-out process. `pow` and `sqrt` are handy little functions that make life easier, all with the help of the `<cmath>` library.

18.1.3 Modulo

The modulo operator (the percent sign: `%`) finds the remainder, or what was left over from division. This program uses the modulo operator to find all prime numbers (all the numbers that never have a remainder of 0 when divided by every number except 1 and itself) that can be held by an `int`.

```
#include <iostream>
using namespace std;
int main()
{
  int testprime = 0, divby = 0, remainder = 0;
  bool isprime;
  cout << "Prime Number Finder" << endl;
  while(testprime < 2147483647)//The Maximum for int
  {
```

```
        isprime=true;
        testprime++;
        for(divby=2; divby < testprime; divby++)
        {
            remainder = testprime \% divby; // store the
                 remainder of the current number when
                divided by divby
            if (remainder == 0)//If the number is not
                prime
            {
                isprime = false;
    break;
            }
        }
        if (isprime)//If it passes the test, it is prime
            .
        {
            cout << " " << testprime; // tell us what
                the prime number is.
        }
    }
    return 0;
}
```

18.2 Review Questions

1. Which `#include` library is needed to use advance arithmetic operators?

2. Write C++ code to calculate 2^9.

3. Write a statement to set the value of a variable of type `double` to the square root of 10001.

18.3 Homework Questions

Complete the code below to find the length of the hypotenuse of a right triangle (remember that $a^2 + b^2 = c^2$) given the lengths of the other two sides. What will be the final output of the code?

```
#include <iostream>
// Add necessary libraries here

using namespace std;

int main()
{
  float a = 3.0, b = 4.0;
  float c;

  //
  // Finish the program...
  //
  cout << "The hypotenuse of the right triangle is "
       << c << endl;
}
```

18.4 Review Answers

- `#include <cmath>` must be included to include advanced operators.
- `pow(2, 9)`
- `double b = sqrt(10001);`

18.5 Homework Answers

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  float a=3.0, b=4.0;
  double c;

  a=pow(a,2);
  b=pow(b,2);
  c=sqrt(a+b);

  cout << "The hypotenuse of the right triangle is "
       << c << endl;
}
```

The final output of the code is:

```
The hypotenuse of the right triangle is 5
```

18.6 Further Reading

- http://pages.cpsc.ucalgary.ca/~jacob/Courses/Fall00/CPSC231/Slides/08-Arithmetic.pdf
- http://www.cplusplus.com/reference/cmath/

Chapter 19

File I/O

Chapter 20

Arrays

Array

Chapter 21

Pointers

Chapter 22

Dynamic Data

Chapter 23

Separate compilation

Chapter 24

Classes and Abstraction

Chapter 25

STL

The Standard Template Library (STL) provides a set of tools beyond those that are provided by the "base" C++ language. In fact, you have already become familiar with some of them [back in Chapter X], such as input and output streams, Y, and Z. While a comprehensive discussion of the features of the STL is far beyond the scope of this text, there are several libraries that offer extremely important features with which you should become comfortable. [footnote: A reasonable place to start for a wider view of the STL is here: Wikipedia Standard Template Library

#include <cstdlib>

#include <vector>

#include <map>

This library provides one of the STL's associative container object classes. An associative container differs from an array in that items in an array are referenced with a number which indicates the item's position in memory:

```
int myArray[10]; // An array of ten integers
myArray[0] = -5; // Set the first integer in the array
     to -5
```

An associative container, on the other hand, can use any data type to reference the items in the container. For example, you might choose to use a string to reference a collection of int items.

```
std::map<std::string,int> myMap;}
```

Perhaps you want to create the object with some initial values:

```
std::map<std::string,int> myMa=p;

// map::at

int main ()
{
 std::map<std::string,int> mymap = {
              { "alpha", 0 },
              { "beta", 0 },
              { "gamma", 0 } };

mymap.at("alpha") = 10;
mymap.at("beta") = 20;
mymap.at("gamma") = 30;

for (auto\& x: mymap) {
  std::cout << x.first << ": " << x.second << '\\n'
      ;
}

return 0;
}
```

The container can only hold one instance of each key, and attempts to add a duplicate item will not overwrite the existing item. The multimap object is a similar structure to map that allows for duplicate items to be added.

Listings