Advanced Arithmetic

***What is it?***

Advanced arithmetic in C++ would be math that can’t be used in regular coding WITHOUT the use of the *<cmath>* library. This is math that goes above and beyond the call of normal duty (which is addition [+], subtraction [-], multiplication [\*], and division [/]). We have already discussed what it is like when a line of code runs through a basic line of math such as:

*int x;*

*x=0;*

*x += 5;*

***return*** *x;*

We have already gone over, but in brief, after we initialize the *x* variable, we set its value to 0. We then take *x*, add 5 to the preexisting value (0), then reset *x* to that new value (5). After that, we return *x* to review the number at the end of the code.

Doing simple operations such as this do not require any special kinds of operational libraries or unique commands. Any compiler can look at a +, -, \*, or / in a line of code and know exactly what the code is asking to have happen. There isn’t any special voodoo who-do, just face value 3rd grade arithmetic. This isn’t always the case though. Sometimes you need to call in the Witch Doctor to accomplish harder math in C++. In this case, our Witch Doctor is the *<cmath>* library.

*<cmath>* is a library that is needed for trigonometric functions, hyperbolic functions, exponential and logarithmic functions, power functions, and rounding or absolute value functions. The *<cmath>* library is designed with the goal in mind of making your life simple and to ease up on the complications that can be brought around with C++. Using the *<cmath>* library in code is as simple as declaring it at the top of your main.cpp file with the rest of your libraries:

*#include <cmath>*

After the declaration of the *<cmath>* library, you can use certain mathematical functions in your code such as power (*pow*), square root (*sqrt*), exponential (*exp*), and others. In early C++ coding, using some of the more advanced mathematical functions that are included in the *<cmath>* library will be fairly uncommon, but for a full list of the mathematical functions, visit this [reference](http://www.cplusplus.com/reference/cmath/)[[1]](#footnote-1). For the purpose of discussion, we will use the *pow* and the *sqrt* mathematical operations and show a few examples.

***Examples:***

*Pow* is the operational function that is called when a line or a section of code needs to be raised to a certain power. Take a look at this code and we’ll break it down line by line to get a grasp of the voodoo that the Witch Doctor is helping with.

*int x, y;*

*x = 4;*

*y = pow (x+1, 3) + 6;*

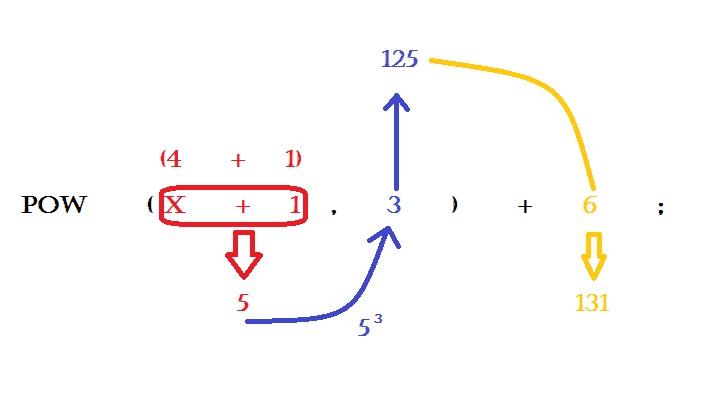
*return y;*

First off, we are initializing two variables, *x* and *y*; nothing that requires voodoo magic just yet. After that, we take *x* and set its value to 4. So far we’re doing pretty well. Now we get to the interesting section of code. We are asking the compiler to raise the value of *x* to the power of 3 plus 1, adding 6, and then setting the whole value to *y*. Now we are in trouble! It’s time to call in the Witch Doctor and get some help. For the use of the *pow* operation, special syntax is required, otherwise there will be compiler errors and a fair bit of headache. 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 break out of the *pow* operation and return to using normal math when we add 6 to 125 and get a value of 131. The final line has us return this value and close the program.

This diagram should help to visualize how the compiler reacts to a *pow* operation when it is encountered in a program:



Undoubtedly there are more complicated uses of the *pow* operation, such as multiple uses of *pow* in the same line of code. You would use multiple *pow* operations in code if a certain formula requires it to be used that way, such as a Pythagorean Theorem function. 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 *z = 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* [so 4]), we raise each value to the power of 2. For visual reference,

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

gets changed to

*z = 32 + 42;*

which is the same as

*z = 9 + 16;*

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

The correlation to exponentswould obviously be square roots. Square root operations are called into the program with *sqrt*. Take a look at the example below to see how it is called in a program:

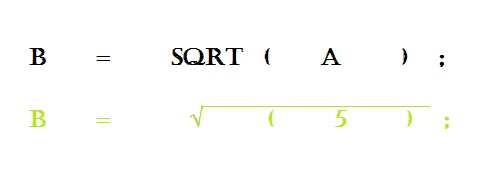
*int a, b;*

*a = 25;*

*b=sqrt (a);*

*return b;*

*sqrt* is simpler than *pow* in that it doesn’t require special syntax in its calling, only another set of parenthesis. More often than not, it is advised to *float* or *double* the value that you will be setting a square root to, but in this example, 25 roots down to a clean 5, so it was *int*ed. For visual reference, refer to the diagram below:



There are cases where both *sqrt* and *pow* are used in the same formula, such as distance between two points on a triangle or other similar equations. When writing such formulas into code, it is very important to keep track of the parenthesis and to use correct syntax in code. One such syntax mistake that is seen a lot is 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 syntactically accurate. When the compiler sees this in code, it doesn’t view it as multiplication, but instead as (in the words of a professional), “function shenanigans.” It is important to be explicit with symbols relating to math in C++. So instead of the syntactically 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 triangle. 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;*

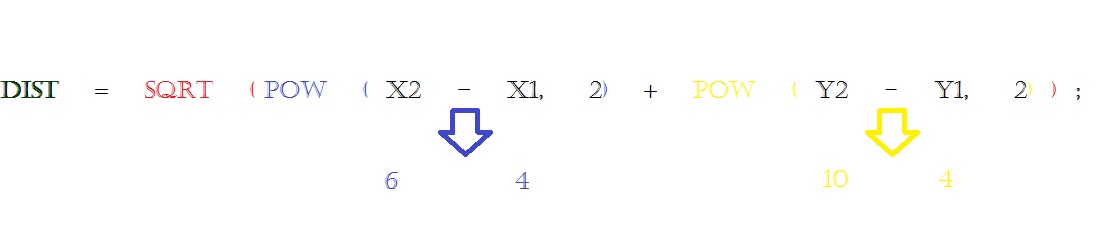
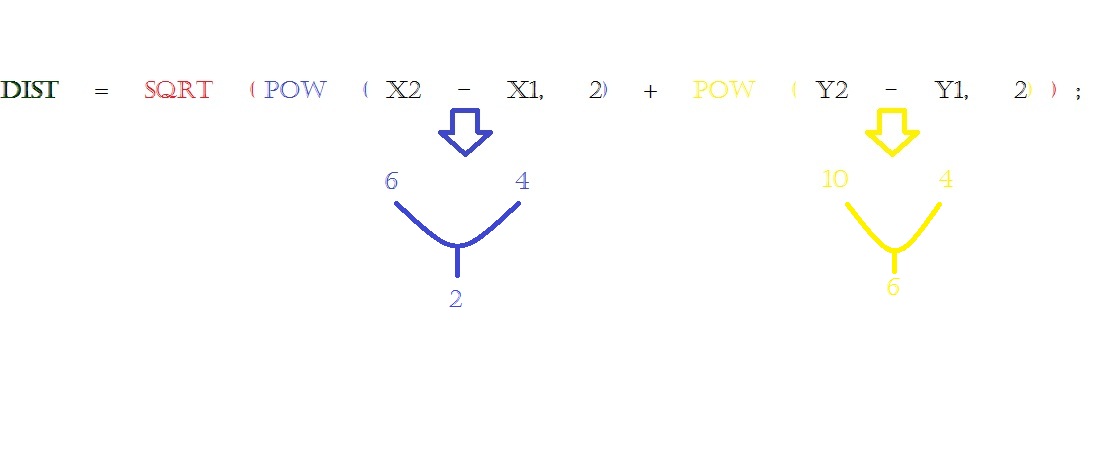
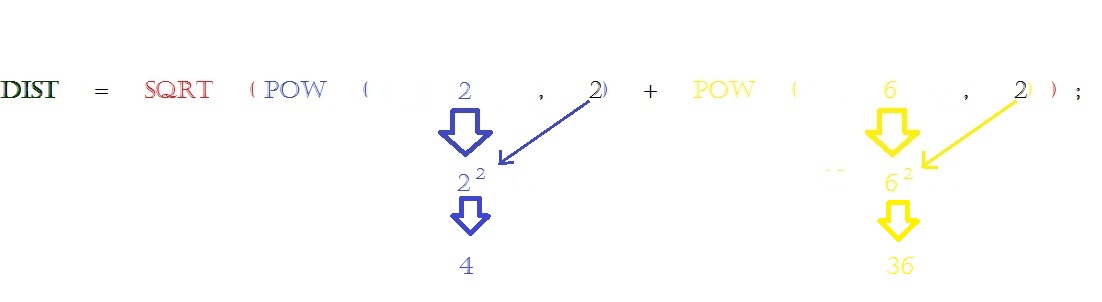
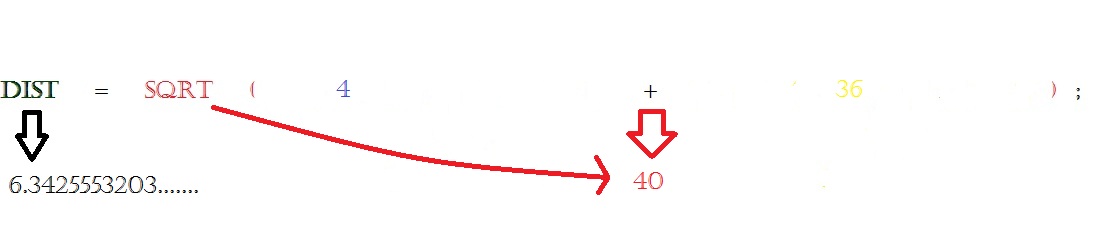
*x2 = 6;*

*y2 = 10;*

*dist = sqrt (pow (x2 – x1, 2) + pow (y2-y1, 2));*

*cout << dist;*

Below is the diagram and the answer to the above problem. For the sake of saving space, the diagram will only include a breakdown of the distance formula.



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

1. http://www.cplusplus.com/reference/cmath/ [↑](#footnote-ref-1)