04. Floating Point Numbers

Objectives

- Use floating point literals
- Use floating point operators
- Declare floating point variables
- Declare and initialize floating point variables
- Use assignment operator for floating variables
- Use type casting
- Understand automatic type conversion and type promotion

If you've finished studying the set of notes on integers (both integer values and integer variables) this one should be easy.

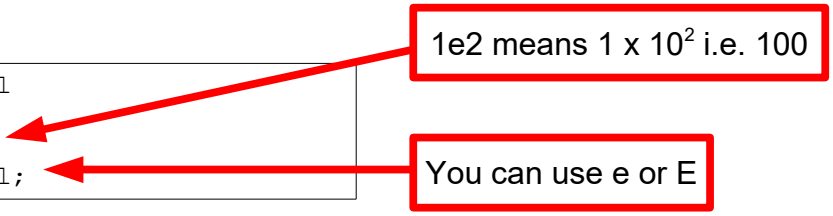
Floats

No, not the A&W kind.

A floating point number is just a number with decimal places (well ... almost). I usually call them floats.

Run this program:

```
std::cout << 1.2345 << std::endl
          << 1e-1 << std::endl
          << 1e2 << std::endl
          << 1.2E+3 << std::endl;
```



Phew! It's good to know that C++ can do decimal places ...

Recall that a **type** is just a collection of values. For instance `int` is a type; it includes values such as 0, 5, -3, and 41.

(WARNING: The word “type” has several meanings in Computer Science. The above is one of them. You can usually discern the intended meaning from the context.)

The name of the type of the above values (1.2345,...) is **`double`**.

C++ actually understands two different kinds for floats: **`float`** and **`double`**. Of course mathematically you can have infinitely many decimal places. For instance your math instructor might write this on the board:

3.333333...

But hey you can't put infinitely many values (the three dots ...) into a computer right?

In the first program above 1.2345 is actually a `double`. If you want to use a `float` you write this:

```
std::cout << 1.2345f << std::endl
          << 1e-1f << std::endl
          << 1e2f << std::endl
          << 1.2E+3f << std::endl;
```

But why are there two kinds of floating point numbers, the `double` and the `float`? Good question!!!

You can be more precise with a `double` than a `float`. Furthermore `double`s represent a wider range of value than the `float`s. It's because of this that a `double` value takes up more memory than a `float` and operations with `double`s might be slower than operations with `float`s.

For this course we will be using mainly the `double` type.

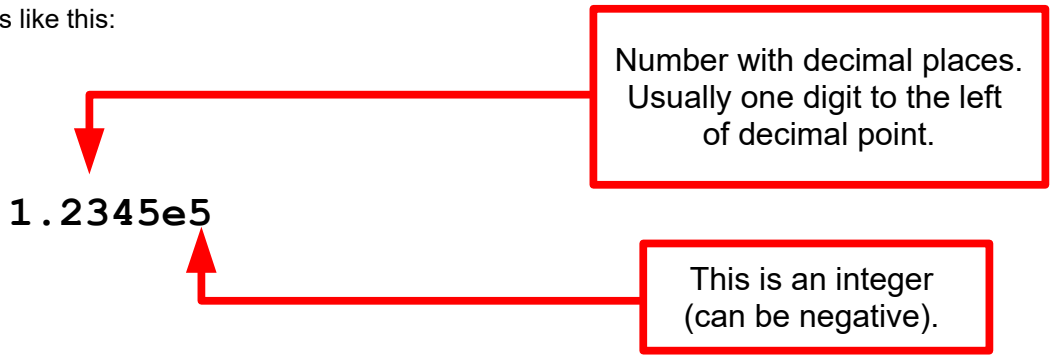
The notation for doubles using e is called the **scientific notation**. Although you can write this in this notation

```
123.45e3
```

it's usually written so it looks like this:

```
1.2345e5
```

So ...



The notation not using e is usually called **fixed-point notation**.

In these notes, to distinguish between the integer five from the floating point number five I will write

- 5 for the integer five
- 5.0 for the floating point five

Exercises. Convert the following to scientific notation. You should check your work with a calculator.

- 12345.0
- -12345.6789
- 0.00001234
- -0.00004564

Exercise. Convert the following numbers from scientific notation to fixed-point notation:

- 1.23E-3
- 123.32e5
- -123.32e-5

Operators

Exercise. Write a program that displays that sum, difference, product and quotient of 1.1 and 2.2.

AHA! You can perform +, –, *, / on floats just like the `int` type.

Exercise. Run this program.

```
std::cout << 1.1 % 2.2 << std::endl;
```

What's the point of this exercise?

Associativity and precedence rules

The associativity and precedence rules are exactly the same as those for integer values (with the caveat that `%` is not defined for floats).

Done! Yeah! :)

Exercise. Which operator is the second to be used for evaluating the following expression:

```
1.234 + (3.4562 + 234.23 * 1.2345) / 1.23
```

Exercise. Which operator is the third to be used for evaluating the following expression:

```
(1.234 - 3.4562 + 234.23) - 1.2345 / 1.23
```

Output: `std::setw()`

The next few sections are some tricks for controlling output.

Exercise. Try the following which has nothing to do with floats:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << "ham" << std::endl;
    std::cout << std::setw(5) << "and" << std::endl;
    std::cout << "eggs" << std::endl;

    return 0;
}
```

Change the 5 to 6 and run the program. Change the 6 to 10 and run it. Change the 6 to 2 and run it. So what does `std::setw()` do? Remove the line `#include <iomanip>` and run it. So what must you remember to do whenever you use `std::setw()`?

An important thing to note is that `std::setw()` only affects one output value. So in the above example, the width for the ***next*** output (i.e. `"eggs"`) is not affected.

Exercise. Change the string `"and"` in the above program to your favorite integer (example: maybe 42?) and run your program. Change the integer value to a double (say 3.14). Does `std::setw()` work with integers and doubles?

Try this:

```
std::cout << std::setw(5) << 2
          << std::setw(5) << 3 << std::endl;
          << std::setw(10) << 1 << std::endl;
```

(And what must you do whenever you use `std::setw()`?) This shows you that you can have multiple width controls in a single `std::cout` statement.

Exercise. Write a program that produces the following output. Do not use the space or the tab character.

```
    n  n^3
    -  ---
    0    0
    1    1
    2    8
    3   27
```

Exercise. Can the integer value used in `std::setw()` be an integer ***variable*** instead? Write a program that works as follows.

```
#include <iostream>
#include <iomanip>

int main()
{
    int w = 0;
    std::cout << "width: ";
    std::cin >> w;
    std::cout << std::setw(w) << 1 << std::endl;

    return 0;
}
```

Run this program entering different values for w.

Exercise. What happens when the width set is too small for the value to be printed? Does C++ chop off (truncate) the value so that it fits? Do this experiment: set the width to 3 and print `"hello"`.

Output: `std::setfill()`

The next set of exercises have to do with `std::setfill()`. Remember that the h character is written `'h'`.

Exercise. Run this program:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::setfill('*')
              << std::setw(3) << 1 << '\n'
              << std::setw(5) << 1 << std::endl;

    std::cout << std::setfill('$')
              << std::setw(3) << 1 << '\n'
              << std::setw(5) << 1 << std::endl;

    std::cout << std::setfill(' ')
              << std::setw(3) << 1 << '\n'
              << std::setw(5) << 1 << std::endl;

    return 0;
}
```

What does `std::setfill()` do? (Duh)

Exercise. Write a program that produces the following output. You are only allowed to press the # key once in your program!!!

```
##########
##########
##########
##########
```

Output: justification

Try this:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::left
              << std::setw(3) << 1 << '\n'
              << std::setw(5) << 1 << std::endl;

    return 0;
}
```

Now try this:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::left
              << std::setw(3) << 1 << '\n'
              << std::setw(5) << 1 << std::endl;

    std::cout << std::right
              << std::setw(3) << 1 << '\n'
              << std::setw(5) << 1 << std::endl;

    return 0;
}
```

Enough said.

Notice that once you do one `std::right` for right justification like this:

```
std::cout << std::right;
          << std::setw(3) << 1 << '\n'
          << std::setw(5) << 1 << std::endl;
```

it will be in effect until you change to left justification.

Exercise. Run this program

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::left;
    std::cout << '<' << 1 << '>' << std::endl;
    std::cout << std::right;
    std::cout << '<' << 1 << '>' << std::endl;
```

```
    return 0;
}
```

and you get

```
<1>
<1>
```

Insert two `<< std::setw(10)` into the code so the you get the following output:

```
<1         >
<          1>
```

Here's another example. Run this program and study the code and the output yourself:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::left << std::setw(12) << "January"
              << '$'
              << std::right << std::setw(10) << 123.45
              << std::endl;

    std::cout << std::left << std::setw(12) << "February"
              << '$'
              << std::right << std::setw(10) << 23.45
              << std::endl;

    std::cout << std::left << std::setw(12) << "March"
              << '$'
              << std::right << std::setw(10) << 3.45
              << std::endl;

    return 0;
}
```

Exercise. Modify the program

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::left << std::setw(12) << "January"
              << '$'
              << std::right << std::setw(10) << 123.45
              << std::endl;

    std::cout << std::left << std::setw(12) << "February"
              << '$'
              << std::right << std::setw(10) << 23.45
              << std::endl;

    std::cout << std::left << std::setw(12) << "March"
              << '$'
```

```
              << std::right << std::setw(10) << 3.45
              << std::endl;

    return 0;
}
```

so that you get this output:

```
January     $....123.45
February    $.....23.45
March       $......3.45
```

Output: `std::fixed` and `std::scientific`

Because there are two formats for `double`s (example: 123.456 is the same as 1.23456e2) sometimes you want to force C++ to choose a particular format.

Try this:

```
#include <iostream>
#include <iomanip>

int main()
{
      std::cout << 0.00001 << std::endl;
      std::cout << 0.00002 << std::endl;

      std::cout << std::fixed;
      std::cout << 0.00001 << std::endl;
      std::cout << 0.00002 << std::endl;

      std::cout << std::scientific;
      std::cout << 0.00001 << std::endl;
      std::cout << 0.00002 << std::endl;

      return 0;
}
```

Output: `std::setprecision()`

First run this:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::setprecision(3);
    std::cout << std::setw(10) << 6789 << std::endl;
    std::cout << std::setw(10) << 67.8 << std::endl;
    std::cout << std::setw(10) << 67.89 << std::endl;
    std::cout << std::setw(10) << 678.9 << std::endl;

    return 0;
}
```

And also this:

```
#include <iostream>
#include <iomanip>

int main()
{
    std::cout << std::setprecision(3) << std::fixed;
    std::cout << std::setw(10) << 6789 << std::endl;
    std::cout << std::setw(10) << 67.8 << std::endl;
    std::cout << std::setw(10) << 67.89 << std::endl;
    std::cout << std::setw(10) << 678.9 << std::endl;

    return 0;
}
```

Change the 3 to 6 in the above programs and run the programs again. So ... what does `std::setprecision()` do?

Note that whereas you need to call `std::setw()` for each value displayed, note that calling `std::setprecision()` will affect all the displayed value until you call it again to set the precision to another number.

Exercise. Modify the above example by print different doubles of different number of decimal places.

Exercise. Do the same experiment as above but with printing set to scientific mode.

```
#include <iostream>
#include <iomanip>

int main()
{
```

```
    std::cout << std::setprecision(3)
              << std::scientific;
    std::cout << std::setw(10) << 6789 << std::endl;
    std::cout << std::setw(10) << 67.8 << std::endl;
    std::cout << std::setw(10) << 67.89 << std::endl;
    std::cout << std::setw(10) << 678.9 << std::endl;

    return 0;
}
```

Get it? You might want to modify the precision and the `double`s being printed.

Exercise. Can you use a variable when specifying the precision?

Double variables

Warning: Incoming warm-up ...

Exercise. Declare a `double` variable `x` with an initial value of 0.0. Print the value of `x`. Run the program and make sure it works. Now fill in the blank:

```
_____________ x = _______________;
```

Exercise. Using the previous program, declare a `double` variable called `pi` with an initial value of 3.14159. Print the value of `pi`.

It's always a good practice to initialize your variables. If you do not have a value in mind, the default choice is `0.0`.

Exercise. Declare two `double` variables `x` and `y`, initializing their values to 1.2 and 3.4 respectively. Print their sum, difference, product and quotient. (Yes, I haven't shown you how to get C++ to compute the sum, difference, product and quotient of **`double`** variables. But you do know how to do the sum, difference, product and quotient of **`int`**s. So make a guess.)

Double input (console window)

Exercise. Be brave. You can do it. Write a program that prompts the user for two floating point numbers (use `double`, not `float`) and then print their sum.

That's it for input to floating point variables!!!

Exercise. Complete the program above so that it prompts the user for two `double`s, prompts the user for a display width, prompts the user for the precision, and finally prints the sum with the given width and precision in fixed point format.

Exercise. Write a mileage calculator. Here's an execution of the program.

```
***** Mileage calculator *****
Enter distance travelled (in miles): 1234
Enter gas used (in gallons): 12
Mileage (miles/gallon): 102.833
```

(Make sure you choose good variable names.)

Exercise. Write an hourly wage calculator. Here's an execution of the program.

```
Pay calculator (version 0.1)
Enter hours: 12.3
Enter pay-per-hour (in dollars): 99.88
Pay: $1228.52
```

Exercise. Write a program that compute the area of a right triangle. Make sure the output is in fixed point mode and with 4 decimal places. Here's an execution of the program:

```
Right triangle area calculator !!!
Base: 12.345
Height: 23.456
The area is 144.7822
```

Here's another execution:

```
Right triangle area calculator !!!
Base: 1
Height: 2
The area is 1.000
```

Types

So far you have seen **five** types of values:

The first is the **C-string**. For instance `"hello world"` is a string. The empty string (or null string) is just `""`.

The second type is the **character**. For instance `'h'` is a character, so is `' '` (the space). There are special character such as `'\n'`, `'\t'`, `'\"'`, etc. A C-string is in fact made up of characters.

The third is the **integer**. For instance 41 is an integer. The C++ name of this type is `int`.

The fourth and fifth type are both **floating point types**. Their C++ names are `double` and the `float`.

I haven't given you the C++ names of the **character** and **C-string** type yet. We will talk about them later. But just to give you something to play with, try this program:

```
char x = '$';
char y = '\n';
char s[100] = "Frodo lives!";
char t[100] = "C++ rules!";

std::cout << x << std::endl
          << y << std::endl
          << s << std::endl
          << t << std::endl;
```

Exercise. You are entirely on your own … be brave … write a program that does this when you run it with this input:

```
what's you fav character? a
a? ... not bad
```

and this when you run it with this input:

```
what's you fav character? T
T? ... not bad
```

Exercise. And now do this … write a program that does this when you run it with this input:

```
i'm c++ ... what's your name? arthur
hi arthur
```

and this when you run it with this input:

```
i'm c++ ... what's your name? zaphod
hi zaphod
```

What happens when you entered `zaphod beeblebrox` instead?

Important warning

It's extremely important to remember that floating point numbers are approximations.

Exercise. Run this program.

```
std::cout << std::fixed << std::setprecision(10)
          << 0.1 + 0.1 + 0.1 << std::endl;
```

No surprises. Now increase the precision to 15 and then to 20 and run the program ...

Got it?

The mystery of what's happening here will be revealed in CISS360 (Assembly language and computer systems).

Typecasting

```
std::cout << 5 << std::endl;
```

And try this

```
std::cout << std::setprecision(10)
          << double(5) << std::endl;
```

`double(5)` returns the `double` value of the integer five.

What do I mean by “return”? It's just like evaluating an expression.

Here's another way to do it:

```
std::cout << std::setprecision(10)
          << (double) 5 << std::endl;
```

Exercise. What does this do?

```
std::cout << int(12.89) << std::endl;
std::cout << (int) 12.89 << std::endl;
```

Exercise. Can you type cast the value of a variable? Try this:

```
int i = 42;
double j = 3.14;
std::cout << (double) i << std::endl;
std::cout << (int) j << std::endl;
```

Do you get an error?

Sometimes C++ will automatically do typecasting for you. Run this program:

```
int i = 7.89;
float x = 5;
double y = 42;

std::cout << i << std::endl;
std::cout << std::fixed << x << '\n'
          << y << std::endl;
```

Then there's **type promotion**. I've already told you that there's the integer addition operator and the double addition operator. Try this:

```
std::cout << 2 + 3.1 << std::endl;
```

This is not the “integer-double” addition operator – there is no such operator!

What happens is that 2 is actually converted to the `double` of 2 before the addition. Of course, the addition is the `double` addition operator.

In general if you apply an operator to an integer and a double (or float),

the integer will first be converted to a double (or float) before the operator is evaluated. That's all there is. Remember that the integer is converted to a double and not the other way round. Why? Because C++ will try to be as accurate as possible.

Exercise. Is the value of this expression an `int` or a `double`?

```
1 + 2 - 3 / 4 * 5.1 + 2
```

Exercise. What integer value is first type promoted to a `double`?

```
4 + 2 - 13 / 4 * 5.1 + 2
```

If you have an integer expression and you really want the output to be a `double`, you can typecast the variables. Try this:

```
int i = 1;
int j = 2;
std::cout << i / j << std::endl;
```

You get zero right? Because the / is the integer division operator. Now change it to this:

```
int i = 1;
int j = 2;
std::cout << double(i) / double(j) << std::endl;
```

Run it again.

Exercise. Run this and explain why the output is the same as the previous program.

```
int i = 1;
int j = 2;
std::cout << double(i) / j << std::endl;
```

Exercise. Run this *version*:

```
int i = 1;
int j = 2;
std::cout << double(i / j) << std::endl;
```

Explain! Pay attention: This is a ***very common mistake!!!***

Exercise. Run this:

```
double x;
std::cin >> x;
int i = x + 0.5;
std::cout << x << ',' << i << std::endl;
```

With several inputs for x (example: 3, 3.1, 3.5, 3.8, etc.) What does this program do?

A function quickie

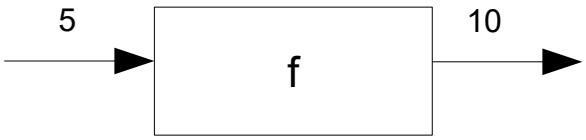
So what is a function? A function in C++ is like a function in math but it's lot more. There is a complete set of notes for functions. Right now I just want to give you enough understanding to move on. Now in math, a function like

$$f(x) = 2x$$

means that

$$f(5) = 2(5) = 10$$

In other words a function in math is something that has an input and an output. For f(x) above, when you put in 5, f(5) gives you 10.



Likewise for the function

$$g(x,y) = xy$$

when you put 4 for x and 7 for y you get

$$g(4,7) = (4)(7) = 28$$

Again the 4 and 7 are the inputs and 28 is the output.

The above examples are functions that performs numeric computations: they take input(s), compute, and return a value. In programming, functions can do more than just computations. Later you will learn that there are functions that open a file for read/write, open a network socket for network communication, open a GUI (graphical user interface) window, etc. Right now I just want to get back to functions that perform numeric computations.

Try this:

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

int main()
{
    std::cout << pow(2.0, -1.0) << '\n'
              << pow(2.0, 1.0) << '\n'
```

```
              << pow(2.0, 2.0) << '\n'
              << pow(2.0, 3.0) << '\n';

    return 0;
}
```

So … what does the function `pow(a, b)` give you?

Note that to use the `pow()` function you must have this in the program:

```
...
#include <cmath>

...
```

Of course you can use expressions:

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

int main()
{
    double x = 2.0, y = -1.0;
    std::cout << pow(x, y) << '\n'
              << pow(x, y + 1.0) << '\n'
              << pow(x * x, y + 5 * x) << '\n';

    return 0;
}
```

WARNING: MS VS .NET will give you an error if you do this:

```
pow(1, 2)
```

For `pow()` to work correctly, at least one of the two values you pass in to the `pow()` function must be a `double`; the return value is a `double`.

Exercise. How would you correct this program without changing the type of `x` and `y`?

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

int main()
{
    int x, y;
    std::cout << pow(x, y) << '\n';

    return 0;
}
```

Exercise. The Pythagoras' theorem states if x, y, z are the lengths of a right angle triangle where z is the hypothenuse, then

$$x^2 + y^2 = z^2$$

Write a program that prompts the user for x and y, and print the value of z.

The Pythagoras' theorem can be used to compute the distance between two points. This can then be used in, for instance, computing if two things collide in a computer game.

Now, you know that `x * x` is the same as `pow(x, 2)`. However you should not overuse `pow()` since the evaluation of `x * x` is probably much faster than `pow()` especially if you're trying to compute the square of integers.

Exercise. Write a program that prompts the user for a `double` and prints the square root of the `double`.

Exercise. Here the famous simple interest formula: If you have P in a savings account at a bank and the bank gives you an interest rate of r (for instance r might be 0.015, i.e. 1.5%), then after t years you will have

$$P(1 + r)^t$$

(P is called the "principal".) Write a program that computes this. Here's an execution of the program:

```
Savings account calculator!!!
Enter P (principal): $1000
Enter r (interest rate in percent): 1.5
Enter t (number of years): 30
Final balance: $1563.08
```

Here's another execution:

```
Savings account calculator!!!
Enter P (principal): $2000
Enter r (interest rate in percent): 1.75
Enter t (number of years): 40
Final balance: $4003.19
```

A bunch of functions available in C++

There are many functions available in C++. Here are some (and don't forget to use `#include <cmath>` when you want to use any of these functions.)

`pow(x, y)` Returns the power of x to y i.e. x^y
x and y are `double`s and the return value is a `double`

`sqrt(x)` Returns the square root of x
x is a `double`, the return value is a `double`

`exp(x)` Returns the power of the natural number e to x, e^x. Here e = 2.718...
x is a `double`, the return value is a `double`

`log(x)` Returns the natural logarithm of x, $\log_e(x)$ which is also written ln(x) in math
x is a `double`, the return value is a `double`

`log10(x)` Returns the logarithm of x to base 10, $\log_{10}(x)$
x is a `double`, the return value is a `double`

`sin(x)` Returns the sine of x, sin(x) where x is in radians
x is a `double`, the return value is a `double`

`cos(x)` Returns the cosine of x, cos(x) where x is in radians
x is a `double`, the return value is a `double`

`tan(x)` Returns the tangent of x, tan(x) where x is in radians
x is a `double`, the return value is a `double`

For all the above functions you can also pass in `float`s instead of `doubles`. If you pass in `float`s the return value is usually a `float`.

(The above list of functions is not exhaustive.)

Exercise. You should know from math classes that

$$\log_{10}(10^x) = x$$

Verify this for the case x = 3 and x = 4.5 using C++.

Exercise. Another version of the Pythagoras' theorem says that

$$\sin^2(x) + \cos^2(x) = 1$$

[Here $\sin^2(x)$ means $(\sin(x))^2$.] Verify this for the case x = 1 and x = 2.34 using C++.

Summary

There are two floating point types: `double` and `float`. A double is capable of more precision than a float.

The following are binary operators available for floating point types (both double and float): +, -, *, /. Note that % is not defined for floating point types. The same precedence and associative rules for integers hold.

The declaration of double or float variables follows the same rules as that or integer variables.

Keyboard input of double variables is similar to integer variables.

`double()` returns the double of a value. `int()` returns the integer of a value.

Assignment and initialization operator (i.e. =) can be used between integers and doubles. For instance you can assign an integer variable a `double` value. An automatic type casting occurs.

When a binary operator operates on an integer and a double, the integer is replaced by its double (it's automatically type casted to a double) before the operator is evaluated.

Exercises

Q1. Write a console program that does the following. It prompts for the user's height, weight, length of his/her thumb, radius of his/her skull and the number college classes the he/she has taken and displays the user's IQ using the following formula

$$iq = PI * skullRadius^2 + \frac{weight}{height + thumbLength} \; x\ classes$$

Your program should use the following for variable names.

- height
- weight
- thumbLength
- skullRadius
- classes
- iq

PI is the constant 3.14159. Your code must of course contain the definition of PI.

Q2. What's wrong with this program:

```
std::cout << "Area of triangle\n";

double height = 0.0; base = 0.0;

std::cout << "Base: ";
std::cin >> based;

std::cout << "Height: ";
std::cin >> height;

std::cout << "Area of triangle with height "
          << base << " and base " << height
          << " is "
          << (1 / 2) * base * height << std::end;
```

Correct it by hand. Verify with C++.

Q3. Do you get the same output for the two print statements?

```
std::cout << (1.2 + 3.4 + 5.6) / 3 << std::endl;
std::cout << (1.2 + 3.4 + 5.6) / 3.0 << std::endl;
```

Verify with C++

Q4. The following should print the average of three integers but does not. Correct it by hand.

```
int x = 0, y = 0, z = 0;
std::cin >> x >> y >> z >> std::endl;
std::cout << (x + y + z) / 3 << std::endl;
```

Verify with C++.

Q5. The formula for converting temperature in Fahrenheit to Celsius is given by this formula:

$$c = \frac{5}{9}(f - 32)$$

(where f is the temperature in Fahrenheit and c is the temperature in Celsius.) Here's an execution of the program:

```
35
1.66667
```

(Of course the input on the first line is the temperature in Fahrenheit and the second line gives the temperature in Celsius.)

Q6. Write a program that does the following:

```
Differentiator!!!
Only for monomials right now ... :(
Enter c and a to differentiate c * x^a: 1.2 5.6

 d
-- (1.2 * x ^ 5.6) = 6.72 * x ^ 4.6
dx
```

Mathematically,

$$\frac{d}{dx}(c\,x^a) = (ca)\,x^{a-1}$$

You do not need to know calculus to do this problem.