03. Integer Variables

Objectives

- Choose valid and appropriate names for variables
- Declare integer variable without initialization
- Declare integer variable with initialization
- Print the value of an integer variable
- Using assignment operator to assign an integer value to an integer variable.
- Assign an integer value entered from the keyboard to an integer variable
- Use integer operators with integer variables

In this set of notes, we will learn to use variables. We will start with integer variables. Once you know how to use integer variables, variables of other types should be easy.

Variables!

We're now ready to put integer values into variables.

The concept of a variable in C++ is similar to the notion of a variable from your math classes.

Once again instead of writing

```
// Name: John Doe
// File: hameggs.cpp

#include <iostream>

int main()
{
    std::cout << "ham\n";
    std::cout << "eggs\n";

    return 0;
}
```

I will write

```
std::cout << "ham\n";
std::cout << "eggs\n";
```

Creating and printing an integer variable

Run this program:

```
int x;
x = 42;
std::cout << x << std::endl;
```

This statement **declares** an integer variable called x

This statement **assigns** 42 to variable x. In this context = is called the **assignment operator**.

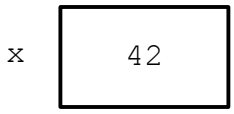
Modify the above program to get this:

```
int x = 42;
std::cout << x << std::endl;
```

Run it.

This statement **declares** an integer variable called x and **initializes** it with a value of 42

This is how you should think of the declaration of an integer variable. Think of your program as creating a box that can contain one integer value and giving the box a name. That's all there is to it.



Make sure you see the difference between initialization and assignment.

There are two things you can do with this box called x. You can either

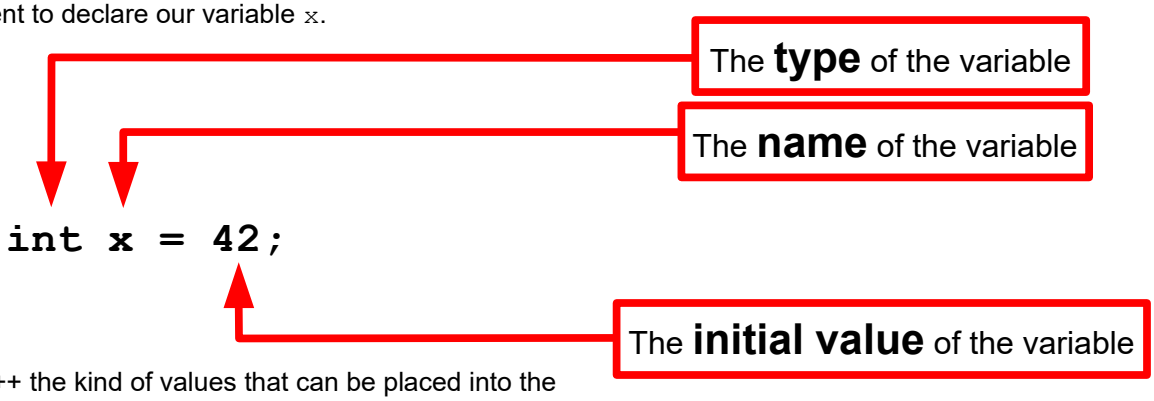
- **read** the value in the box

or you can

- **put** a new value into the box.

When you put a new value in, the old value is wiped out.

Let's look at the statement to declare our variable `x`.



The word **`int`** tells C++ the kind of values that can be placed into the box of `x`. In this case it tells C++ that `x` can hold **integer values**. `int` is a **type**. We'll see more types later.

So the format of the statement to declare a variable is either:

`[type] [variable name];`

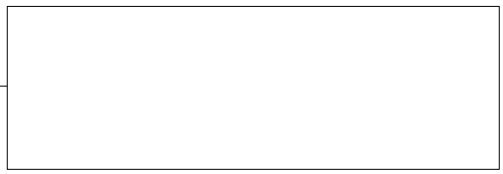
or

[type] [var name] = [init val];

where the second format includes an initialization. If there is an initialization, then the initial value is placed in the box. What if there's no initialization?

Try this:

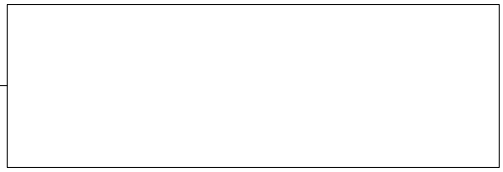
```
int x;
std::cout << x << std::endl;
```



In general, it's a good practice to initialize variables.

Exercise. This won't work:

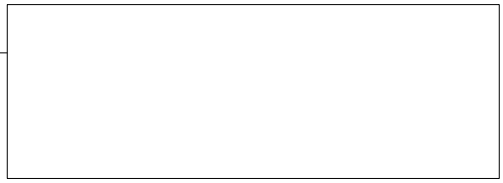
```
std::cout << x << std::endl;
int x = 42;
```



Run it and read the error message. What's the problem?

Exercise. This won't work either:

```
int x = 42;
int x = -1;
```



Try to run it. Read the error message. What's the problem?

You can assign values to your variable as many times as you like. Try this

```
int x = 42;
std::cout << "x = " << x << std::endl;
x = -5;
std::cout << "x = " << x << std::endl;
x = 1356;
std::cout << "x = " << x << std::endl;
```

It's important to note that once you assign a value to `x`, the old value is overwritten. `x` can hold only one value at one time.

Make sure you see the difference between the following two print statements!!!

```
int x = 42;
std::cout << "x" << std::endl;
std::cout << x << std::endl;
```

Make sure you see the difference between the = (initialization) in

```
int x = 42;
```

and the = (assignment) in

```
x = -5;
```

Exercise. Here's an easy warm up. Write a program that does the following:

- Create an integer variable with the name `i` with initial value of 0
- Print the value of `i`.

Exercise. Complete this program by writing two statements. The first statement declares a variable. Here's what you need:

- The type of the variable is `int`.
- The name of the variable is `num_arms`
- The initial value of the variable is 3.

Don't forget the underscore _ in the variable name! The second statement is a print statement. Refer to the print out and you should be able to finish the program. The print statement must involve your variable.

```
std::cout <<
```

When you run the program you should get this output:

```
I have 3 arms.
```

Exercise. 42 is an integer value. You have already seen another type of value: the C-string. Can you initialize an integer variable a C-string value? Try this:

```
int x = "ham";
```

Or how about assignment:

```
int x;
x = "ham";
```

Integer variables and operators

Try this:

```
int x = 42;
std::cout << x + 1 << std::endl;
std::cout << 1 + x << std::endl;
```

Not too surprising, right?

Now let's try something with two variables. (It's not too surprising that you can have two variables – I hope!)

```
int x = 42;
int y = 24;

std::cout << x + y << std::endl;
std::cout << x - y << std::endl;
std::cout << x * y << std::endl;
std::cout << x / y << std::endl;
std::cout << x % y << std::endl;
```

Now look at the above declaration of the variables. How many statements are there?

You can also actually declare ***several*** variables in ***one*** statement:

```
int x = 42, y = 24;

std::cout << x + y << std::endl;
std::cout << x - y << std::endl;
std::cout << x * y << std::endl;
std::cout << x / y << std::endl;
std::cout << x % y << std::endl;
```

You can also choose either to initialize or not initialize the variables you're declaring:

```
int x = 42, y;
y = 24;

std::cout << x + y << std::endl;
std::cout << x - y << std::endl;
std::cout << x * y << std::endl;
std::cout << x / y << std::endl;
std::cout << x % y << std::endl;
```

Exercise. Refer to the previous exercise. Is it possible to initialize the second variable and not the first? Verify yourself with a C++ program!

Exercise. Spot the syntax error(s):

```
int x = 42; y = 24;
std::cout << x + y << std::endl;
```

Exercise. What is the output of this program:

```
int x = 3,y = 2, z = -2;
std::cout << x + y * 3 / x - z * 4 - (2 - y) % x
          << std::endl;
```

The assignment operator

OK. Pay attention to this.

In programming, = is not exactly the same as in Math. In Math, = can be used to denote an **equation**. For instance:

5x = 3 + x

When you're told (in a Math class of course) to “solve this equation”, it means that you are to find “the value(s) of x satisfying the equation 5x = 3 + x”. (Of course in this case the solution for x is 3/4).

In most programming languages, = denotes means **assignment** or **initialization**. When you have a program statement that looks like:

```
x = y + z + 3;
```

it means:

- **Evaluate the expression on the right, and**
- **Give the value on the right to the variable on the left**

Exercise. Using the above recipe for =, “execute” this program by hand.

```
int x = 42;
x = x + 1;
std::cout << x << std::endl;
```

Now verify your work by running the program.

Of course in math an **equation** like

x = x + 1

becomes

0 = 1

which is gibberish. In other words you cannot solve this equation (at least not in real numbers). In C++

```
x = x + 1;
```

makes sense because = is an assignment and has nothing to do with

equations.

So remember this: In math = means equation. In C++ = means assignment or initialization.

Exercise. First figure out the output by hand. Next run the program

```
int x = 2, y = 3;
std::cout << x << ", " << y << std::endl;
x = x + y;
std::cout << x << ", " << y << std::endl;
```

Exercise. True or false: The output of this

```
int x;
2 = x
std::cout << x + 1 << std::endl;
```

is

```
3
```

Check with your C++ compiler.

Exercise. Try this:

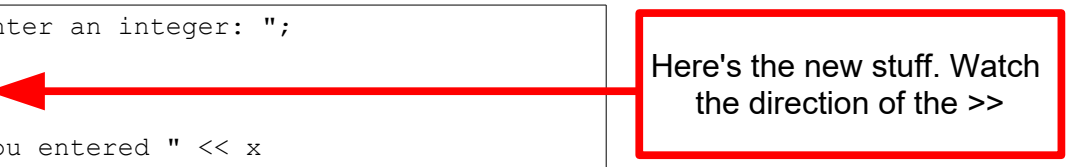
```
int x = 0, y = 0;
x + y = x + 1;
```

Run this. Can you explain why there's an error?

Keyboard input (console window)

Run this program.

```
std::cout << "Enter an integer: ";
int x = 0;
std::cin >> x;

std::cout << "You entered " << x
          << '.' << std::endl;
```



Enter an integer value (example: 42) and press the enter key. Run the program again, entering a different value (example: 167).

Notice that the value of `x` printed is the value entered. And who's the culprit that gave the value entered to x? Clearly it's

```
std::cin >> x;
```

In other words the above statement puts the value entered through the keyboard into the variable `x`.

Get it?

Exercise. Does this work? Run it to verify.

```
std::cout << "Enter an integer: ";
std::cin >> x;
int x = 0;

std::cout << "You entered " << x
          << '.' << std::endl;
```

Why?

Exercise. Write a program that prompts you for your age, and then prints it. The following is an execution of the program when the user enters 15 (the value is in bold).

```
How old are you? 15
You are 15 years old. Of course you could be lying.
```

Here's another execution of the same program where the user enters 142:

```
How old are you? 142
You are 142 years old. Of course you could be lying.
```

Convention: Input data is printed in bold.

Now let's try an example with two inputs:

```
int x = 0;
std::cout << "Enter integer x: ";
std::cin >> x;

int y = 0;
std::cout << "Enter integer y: ";
std::cin >> y;

std::cout << "x + y = " << x + y;
```

Run this twice, entering different values for `x` and `y`. As you run the program, try to match what you see in the console window with the corresponding statement in the program.

Exercise. Write a program that does the following. Here's one execution of the program:

```
Multiplicator !!!
Enter an integer: 5
Enter another integer: 7
5 * 7 = 35
```

Here is another execution of the same program:

```
Multiplicator !!!
Enter an integer: -3
Enter another integer: 4
-3 * 4 = -12
```

Recall that for output you can print several things:

```
std::cout << x << ", " << y << std::endl;
```

It turns out that input is very similar. You can have one input statement that handles more than one variable:

```
int x = 0, y = 0;
std::cout << "Give me x and y: ";
std::cin >> x >> y;
std::cout << x << ", " << y << std::endl;
```

Run it. When you're prompted for x and y, you can either enter the values separated by spaces (1 or more):

```
Give me x and y: 2 5
2, 5
```

Or you can separate the values with tabs (1 or more):

```
Give me x and y: 2	5
2, 5
```

Or you can separate the values with newlines (1 or more, using the Enter key):

```
Give me x and y: 2
5
2, 5
```

Or any combination of spaces, newlines, or tabs:

```
Give me x and y:
                                2




        5
2, 5
```

Basically your program will scan through all the data entered, skipping whitespaces (spaces, tabs, newlines) if necessary, to find the relevant integer values to be placed into variables `x` and `y`.

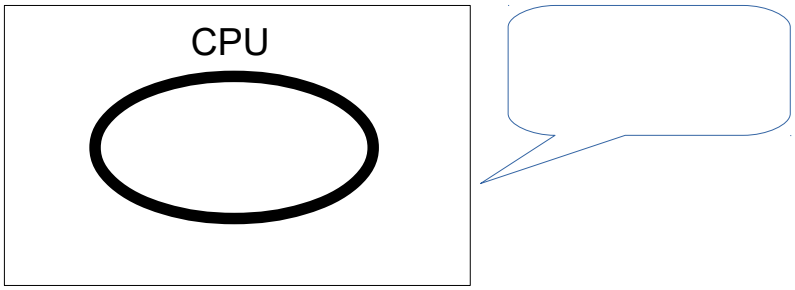
Computational model

Sometimes it's useful to “visualize” the execution of a program. The following is a model of the computer. We will execute a program on it. Remember that **this is only a model**. In fact, later I will need to increase the power of this “computer” because this model is not powerful enough.

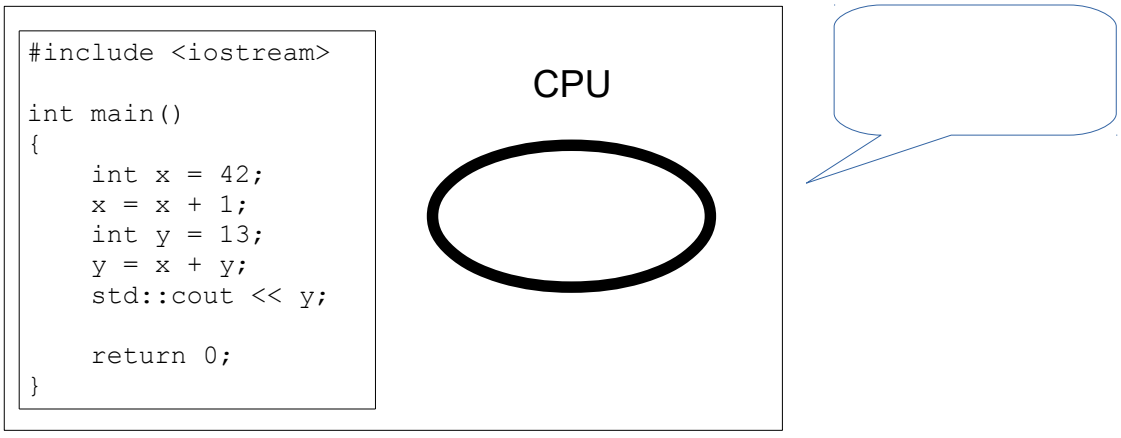
To illustrate the computation in this computer, let's run it on the following program:

```
#include <iostream>

int main()
{
    int x = 42;
    x = x + 1;
    int y = 13;
    y = x + y;
    std::cout << y;

    return 0;
}
```

So here's your computer:



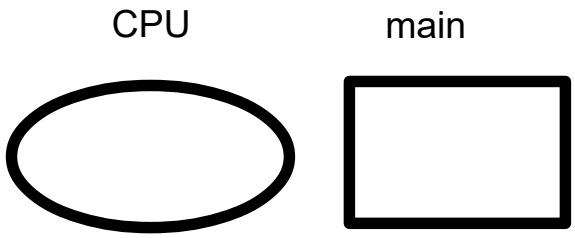
The speech bubble is the output. Everything printed appears in it. The CPU performs math. Now when you run the program, the computer loads the program:



And when it runs, the first thing that happens is that it creates a place for

keeping variables:

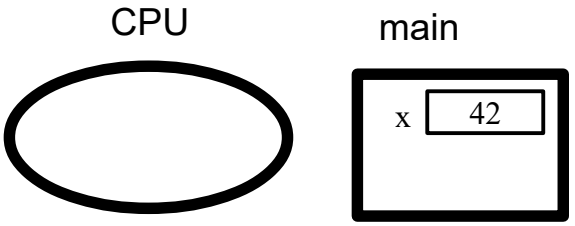
```
#include <iostream>

int main()
{
    int x = 42;
    x = x + 1;
    int y = 13;
    y = x + y;
    std::cout << y;

    return 0;
}
```



You can think of the box named main as an area of putting variables.

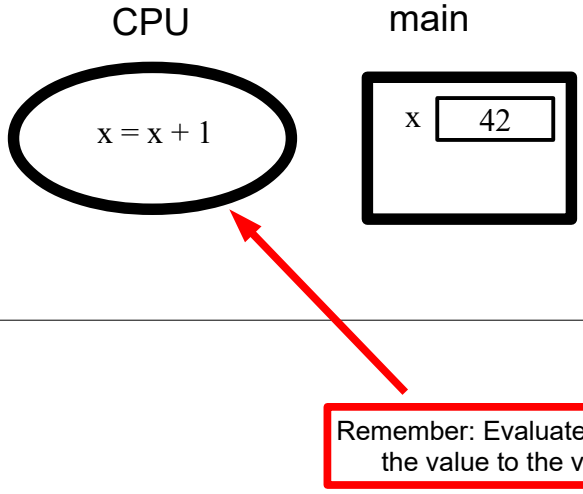
It then goes on to the next statement, a declaration and initialization:

```
#include <iostream>

int main()
{
    int x = 42;
    x = x + 1;
    int y = 13;
    y = x + y;
    std::cout << y;

    return 0;
}
```

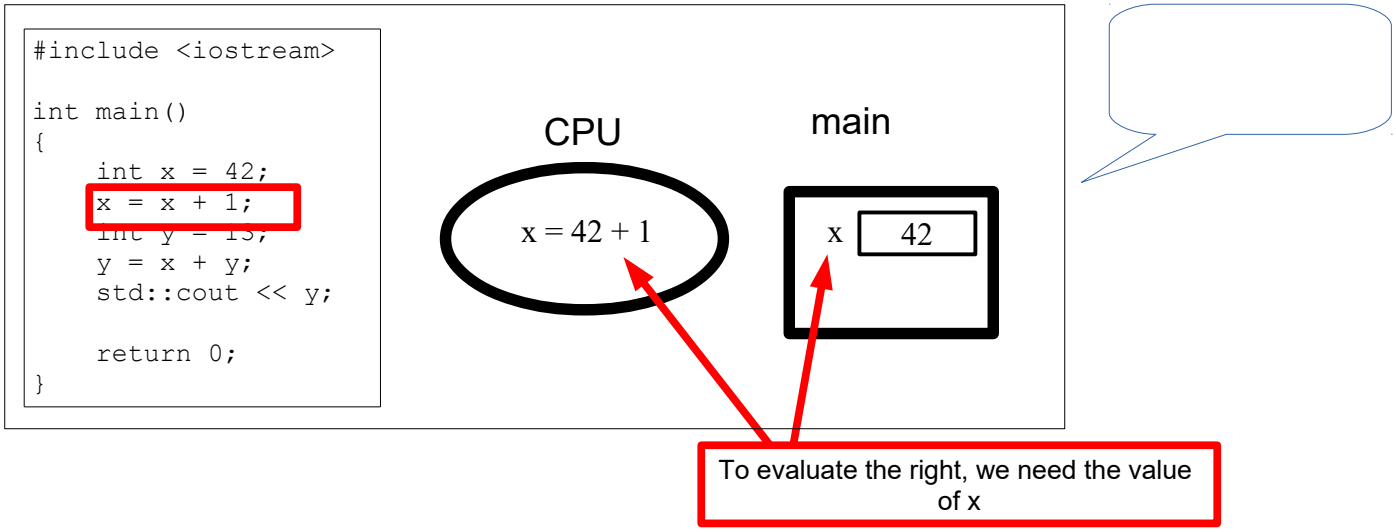


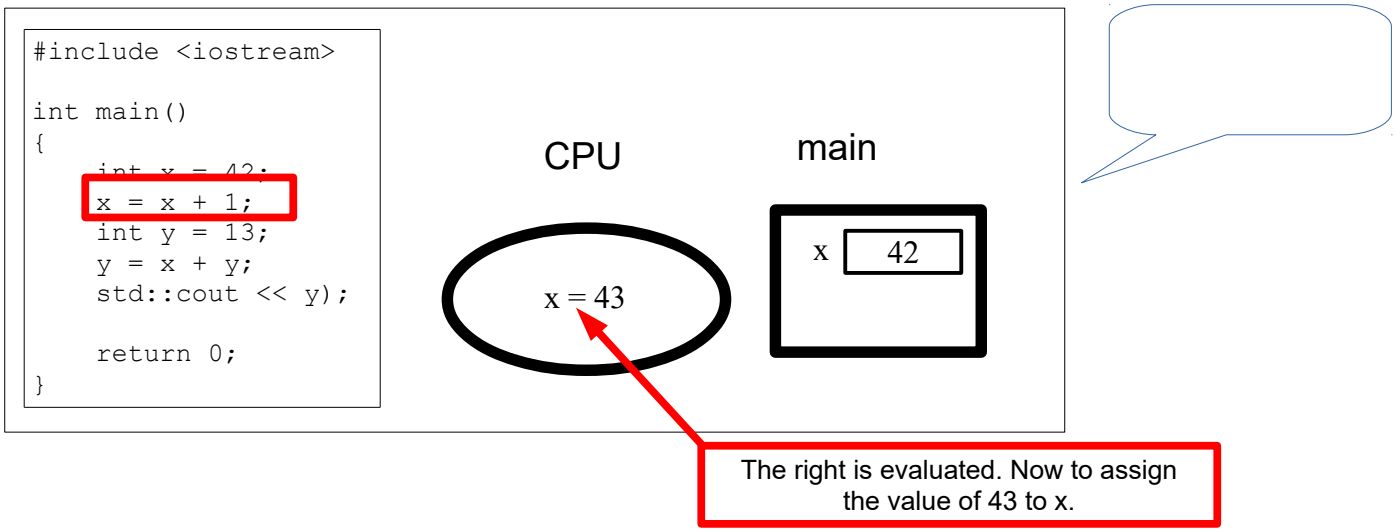
Now for an assignment:

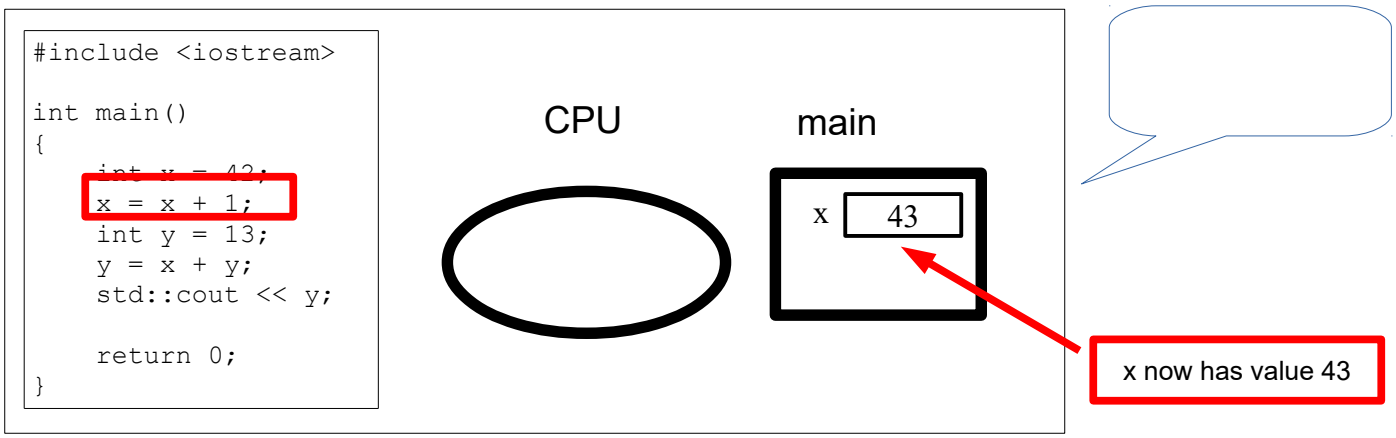
```
#include <iostream>

int main()
{
    int x = 42;
    x = x + 1;
    int y = 13;
    y = x + y;
    std::cout << y;

    return 0;
}
```



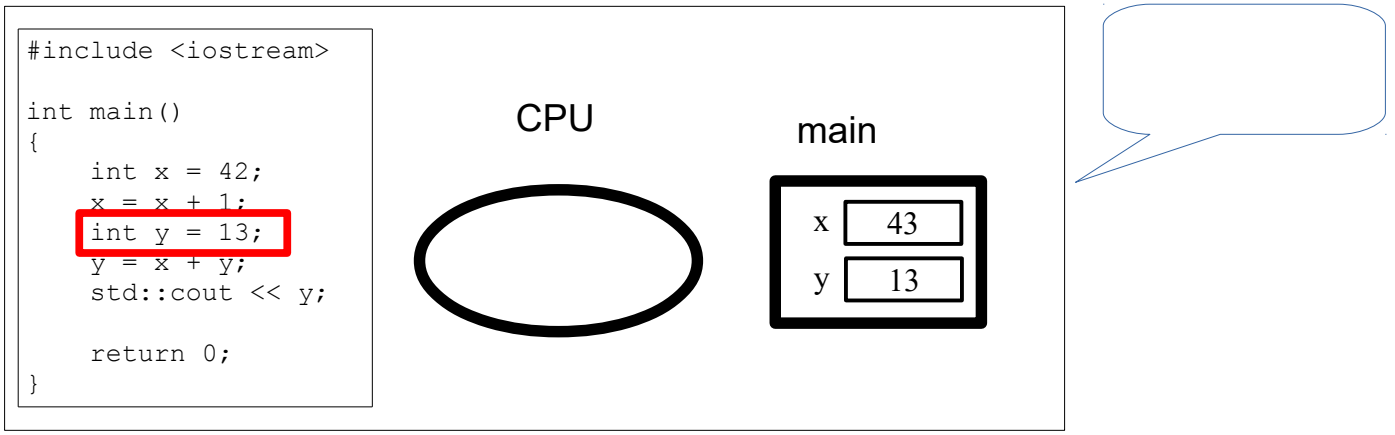
Remember: Evaluate the right and assign the value to the variable on the left


#include <iostream>
int main()
{
int x = 42;
x = x + 1;
int y = 13;
y = x + y;
std::cout << y;
return 0;
}
CPU
main
x = 42 + 1
x
42
To evaluate the right, we need the value of x

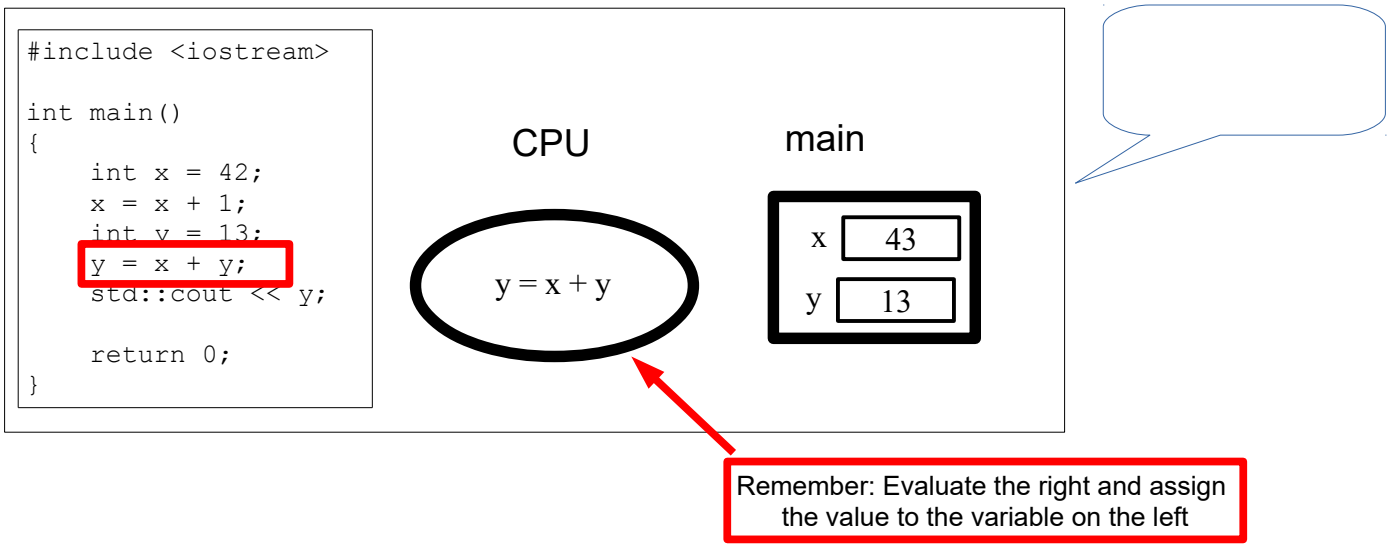

#include <iostream>
int main()
{
int x = 42;
x = x + 1;
int y = 13;
y = x + y;
std::cout << y);
return 0;
}
CPU
main
x = 43
x
42
The right is evaluated. Now to assign the value of 43 to x.

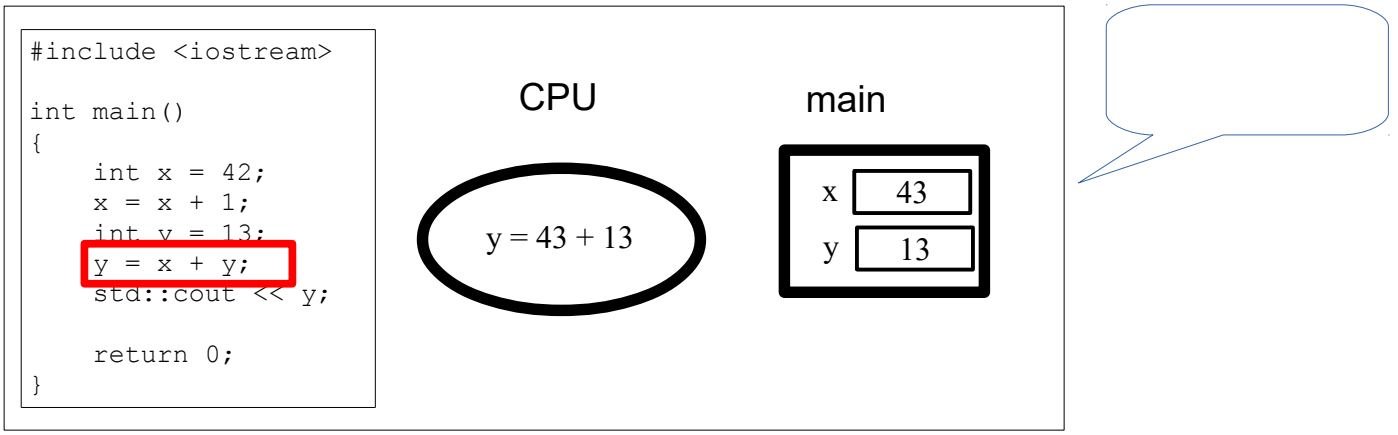

#include <iostream>
int main()
{
int x = 42;
x = x + 1;
int y = 13;
y = x + y;
std::cout << y;
return 0;
}
CPU
main
x
43
x now has value 43

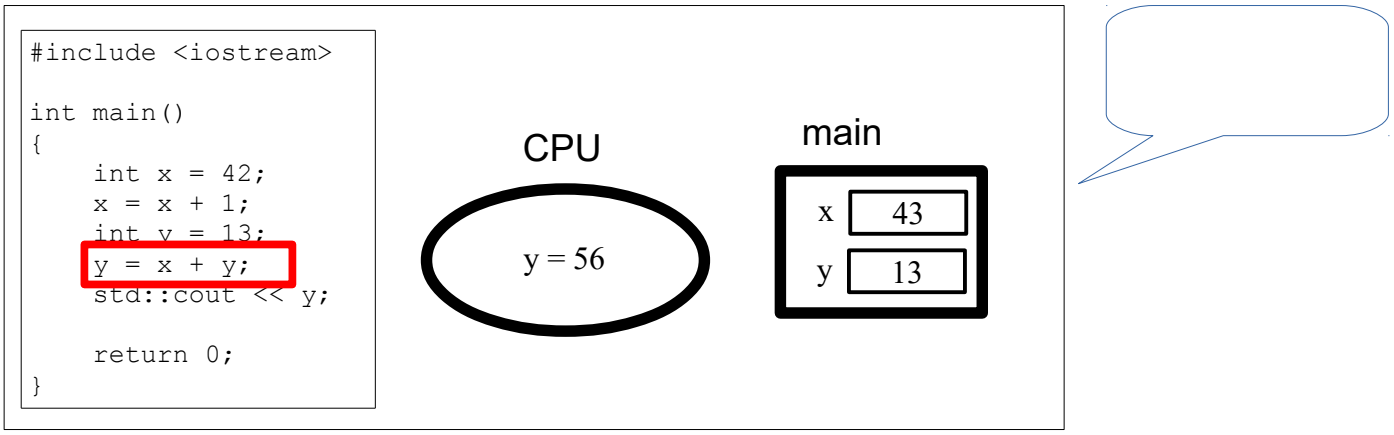
Done! The next statement is the declaration and initialization of y:

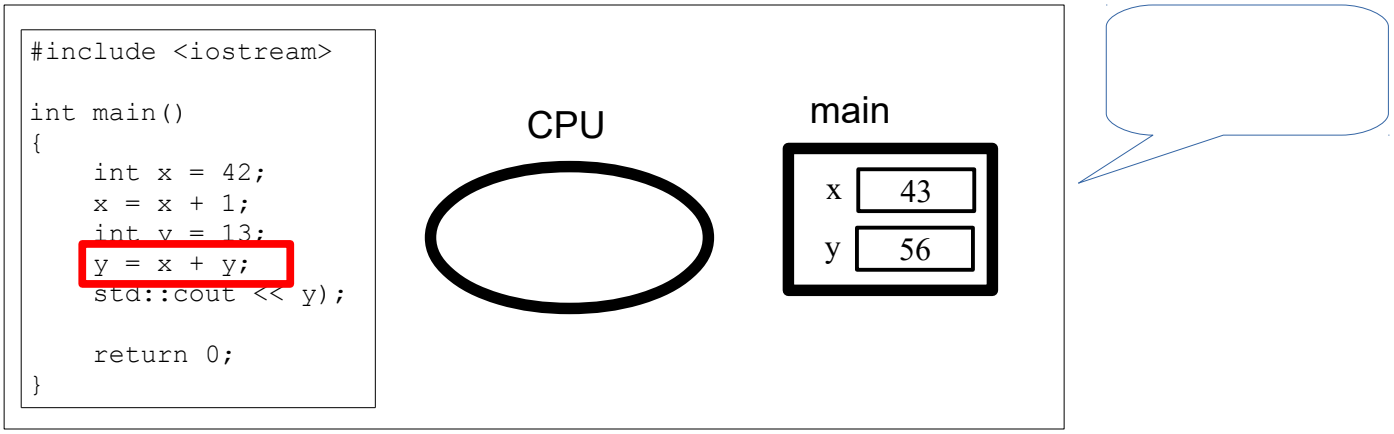


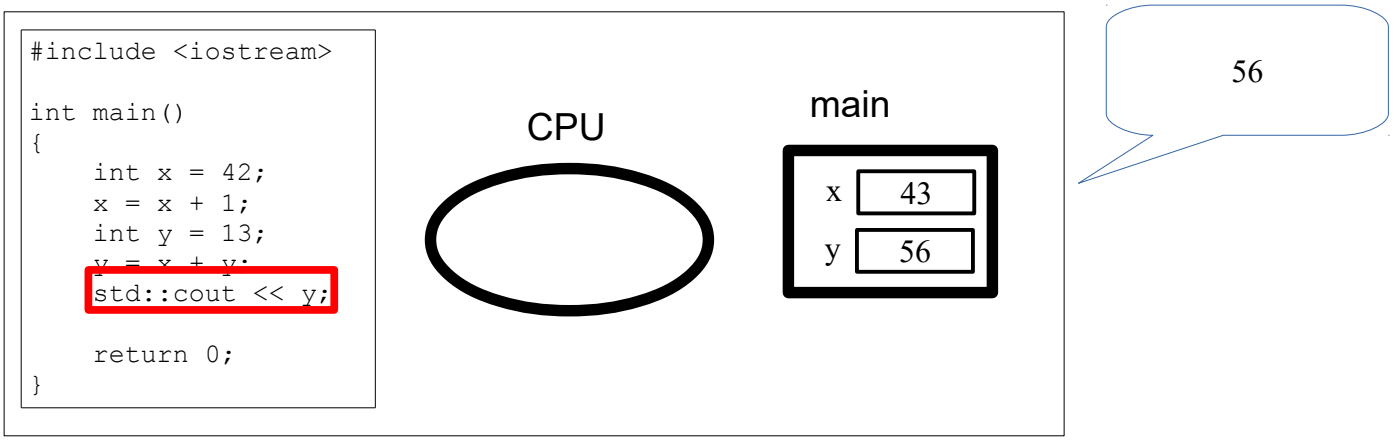
This is followed by another assignment:







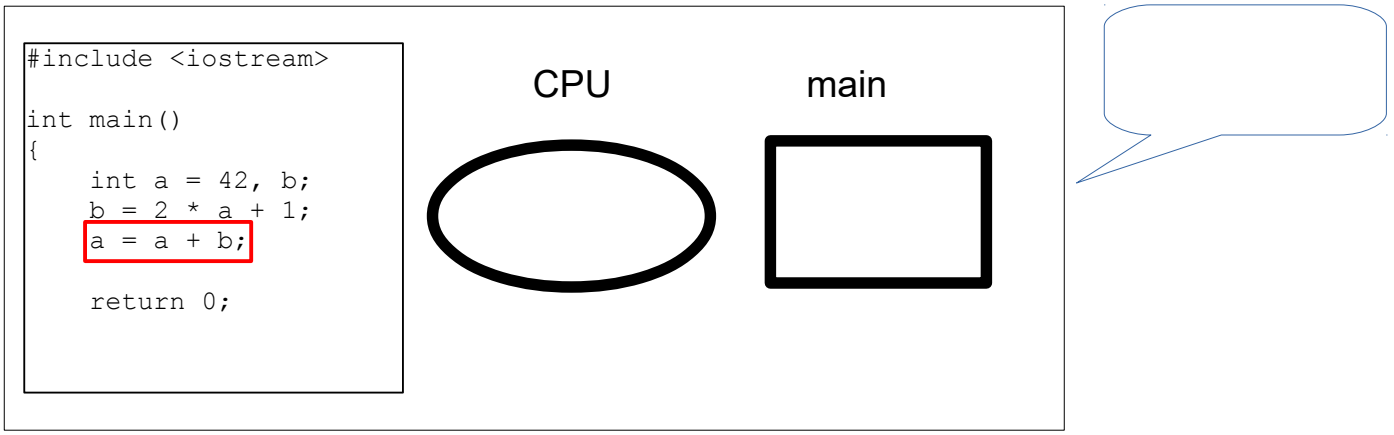




The important thing to remember is that right now the computer executes one statement at a time from top to bottom.

Again, this is only a model.

Exercise. Complete the main box at the point when C++ finish executing the statement in red.



When you're done, you can check if you're correct by adding a statement to print the value of `a` and `b` at the right place and run the program with your C++ compiler to see if you're correct.

Exercise. Take a sheet of paper and perform a trace like the above with the follow code:

```
#include <iostream>

int main()
{
    int a = 4;
    int b = a - 2;
    std::cout << a << ", " << b << std::endl;

    a = b * a;
    std::cout << a << ", " << b << std::endl;

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Run this program with your C++ compiler and compare.

Tracing programs with repeating chunks of statements

The following exercises are simple but extremely important. They all involve tracing programs … with repeating chunks of code.

Exercise. This is an important exercise! Take a sheet of paper and perform a trace like the above with the follow code:

```
#include <iostream>

int main()
{
    int i = 0;

    i = i + 1;
    std::cout << i <<  '\n';

    i = i + 1;
    std::cout << i <<  '\n';

    i = i + 1;
    std::cout << i <<  '\n';

    i = i + 1;
    std::cout << i <<  '\n';

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Next run this program and compare your trace with the program's output.

Exercise. This is an important exercise! Take a sheet of paper and perform a trace like the above with the follow code:

```
#include <iostream>

int main()
{
    int i = 10;

    i = i + 2;
    std::cout << i <<  '\n';

    i = i + 2;
    std::cout << i <<  '\n';

    i = i + 2;
    std::cout << i <<  '\n';

    i = i + 2;
    std::cout << i <<  '\n';

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Next run this program and compare your trace with the program's output.

Exercise. This is an important exercise! Take a sheet of paper and perform a trace like the above with the follow code:

```
#include <iostream>

int main()
{
    int i = 1;

    i = i * 2;
    std::cout << i <<  '\n';

    i = i * 2;
    std::cout << i <<  '\n';

    i = i * 2;
    std::cout << i <<  '\n';

    i = i * 2;
    std::cout << i <<  '\n';

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Next run this program and compare your trace with the program's output.

Exercise. The following is a *very* important exercise!!! Take a sheet of paper and perform a trace like the above with the follow code:

```
#include <iostream>

int main()
{
    int i = 0, s = 0;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Next run this program and compare your trace with the program's output. What has the above got to do with 0 + 1 + 2 + 3 + 4?

Exercise. The following is a *very* important exercise!!! Take a sheet of paper and perform a trace like the above with the follow code. WARNING: This very similar to the above.

```
#include <iostream>

int main()
{
    int i = 0, s = 0;
    std::cout << i << ' ' << s << '\n';

    i = i + 1;
    s = s + i;
    std::cout << i << ' ' << s << '\n';

    i = i + 1;
    s = s + i;
    std::cout << i << ' ' << s << '\n';

    i = i + 1;
    s = s + i;
    std::cout << i << ' ' << s << '\n';

    i = i + 1;
    s = s + i;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Exercise. The following is also a *very* important example! Take a sheet of paper and perform a trace like the above with the follow code:

```
#include <iostream>

int main()
{
    int i = 1, p = 1;
    std::cout << i << ' ' << p << '\n';

    p = p * i;
    i = i + 1;
    std::cout << i << ' ' << p << '\n';

    p = p + i;
    i = i + 1;
    std::cout << i << ' ' << p << '\n';

    p = p + i;
    i = i + 1;
    std::cout << i << ' ' << p << '\n';

    p = p + i;
    i = i + 1;
    std::cout << i << ' ' << p << '\n';

    return 0;
}
```

What is the output? Write it down here using one square for each output character or digit:

Make sure your trace-by-hand matches the output of the program when you run it. What the above got to do with 1 x 2 x 3 x 4? What if I changed the initial value `p` to 0? Redo the trace and see what you get.

Exercise. The following is **also important**. Take a sheet of paper and perform a trace … WAIT!!! Look at the code:

```
#include <iostream>

int main()
{
    int i = 0, x = 0;
    std::cout << i << ' ' << x << '\n';

    i = i + 1;
    x = x + i * i;
    std::cout << i << ' ' << x << '\n';

    i = i + 1;
    x = x + i * i;
    std::cout << i << ' ' << x << '\n';

    i = i + 1;
    x = x + i * i;
    std::cout << i << ' ' << x << '\n';

    i = i + 1;
    x = x + i * i;
    std::cout << i << ' ' << x << '\n';

    return 0;
}
```

Can you very quickly figure out the output without a detail trace? (If you've understood the above exercises completely you should be able to do it. Next, run this program and compare it with your trace. Make sure your trace-by-hand matches the output of the program when you run it.

Note that in the above exercises, a chunk of code is repeated. For instance for this program:

```
#include <iostream>

int main()
{
    int i = 0, s = 0;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    return 0;
}
```

notice that this code segment is repeated:

```
    i = i + 1;
    s = s + i;
    std::cout << i << ' ' << s << '\n';
```

Frequently, after tracing a few chunks of repeated code, once you see the pattern, you can guess the output for the rest.

Exercises. Go through all the examples above and locate repeated code segments. Can you figure out quickly the output without tracing every line slowing?

Rules for creating names

Try this

```
int #@$! = 42;
```

(Don't panic. Don't call 911. You *should* get an error.)

Try this:

```
int num arms = 3;
```

Again you should get an error.

Rules

There are rules that you must follow for creating variable names in C++. If you don't follow them, your C++ compiler will yell at you and refuse to compile and run your code.

- You can only use alphanumerics (i.e. a-z, A-Z, 0-9), underscore (i.e. _).
- The first character of the name cannot be numeric.
- You cannot use a keyword as a name. A **keyword** is just a word that C++ has already taken. For instance the word “int” and “return”, etc. are already used by C++.

Again … these rules MUST be followed otherwise C++ will shout at you.

Common Practices

The following are ***not*** rules but common practices in naming C++ variables. This means that the following are advice and your compiler will probably run your code even if you ignore the advice. However you should still follow the advice because all good programers should.

1. A name can be as long as you like (depending on the compiler). But be reasonable. Something like

   ```
   numberOfSpaceInvadersStillAliveAndWellAndCausingTrouble
   ```

   is hard to read and even harder to type … *correctly!!!*

2. The first character is a lowercase letter. (You'll see exceptions later.)

3. If a variable represents something in the real world you're modeling, you should use a meaningful name for it.

4. It's a convention that if your variable name is made up of words, you should either
   1. capitalize the first letter of each word in the variable name (except for the first word), or
   2. insert an underscore between the words.

   Example:

   ```
   int energylevel = 10000;  // BAD!
   int energyLevel = 10000;  // Good
   ```

```
int energy_level = 10000; // Good
```

In general you choose either style 1 or 2 but not both in the same program.

5. Although you can start with an underscore, you should not. This is because your C++ compiler actually includes extra variables into the final program. These special variables usually have names starting with either one or maybe even two underscores.

6. Of course variable names are case-sensitive:

```
int energylevel = 10000;
int energyLevel = 10000;
```

Make sure you know the difference between rules and advice for choosing variable names.

If you work for a company, you might find that they have their own rules in order to make code more readable according to *their* practices.

Jargon

In programming languages, an **identifier** is just a technical term for "name". So a variable name is an identifier.

You have in fact seen another name: the word `main` is an identifier. I'll come to this later.

Exercise. Which of the following variable names are valid?

- `gpa`
- `_gpa`
- `gradePointAverage`
- `gradepointaverage`
- `greatpointavarag`
- `grade_point_average`
- `grade$point$average`
- `gradePointAverage3`
- `3gradePointAverage`
- `grade Point Average`

Check with your C++ compiler.

(Note that you cannot use $ in variable names. Some programming languages allow it, but not C++.)

Exercise. True or false? The output of this code

```
int num arms = 3;
int num legs = 4;
std::cout << "number of limbs: "
          << num arms + num legs
          << std::endl;
```

is

```
7
```

Exercise. Rename the variables using more descriptive variable names:

```
int x = 0;
std::cout << "Number of martians: ";
std::cin >> x;

int y = 0;
std::cout << "Fingers on each martian: ";
std::cin >> y;

std::cout << "Fingers altogether: " << x * y
          << std::endl;
```

Swapping values in variables

Pay attention!!!

Using the computational model in the notes, trace the following extremely important trick. It "swaps" the values of variables `x` and `y`.

```
#include <iostream>

int main()
{
    int x = 4, y = 2;
    std::cout << x << ", " << y << std::endl;

    int t = x;
    x = y;
    y = t;
    std::cout << x << ", " << y << std::endl;

    return 0;
}
```

Exercise. Complete the following program so that it swaps the variables `x`, `y`, `z` "in a circle":

```
#include <iostream>

int main()
{
    int x = 4, y = 2, z = -6;
    std::cout << x << ' ' << y << ' ' << z
              << std::endl;









    std::cout << x << ' ' << y << ' ' << z
              << std::endl;

    return 0;
}
```

The expected output is

```
-6 4 2
```

Limit On C++ Integer Value

In math, there is no limit on integer values. There's nothing wrong with your math teacher writing

12323324985304534853345

on the board. One thing to watch when writing C++ programs is that C++ integers have limits. On a 32-bit machine, an integer value *usually* ranges from -2^{31} to $2^{31} - 1$, i.e.

-2,147,483,648 to 2,147,483,647

What happens when you go beyond? The numbers “cycles around”. In other words if you declare:

```
int x = 2147483647;
x = x + 1;
std::cout << x << std::endl;
```

you'll find that `x` becomes -2147483648. Yikes!!! Furthermore

```
int y = -2147483648;
y = y - 1;
```

will give y the value `2147483647`.

You know what that means right? If you write a program for a bank and you have the following:

```
int total_assets = 2147483647;
int deposit;
std::cin >> deposit;
total_assets = total_assets + deposit;
```

you could very well end up with a negative number!!!

At this point you need not worry about about how programmers handle this problem (as well as lawsuits). You just need to know that most C++ integers in a 32-bit machine is between roughly -2 billion to 2 billion.

Summary

An integer variable has a name and a space for an integer value. You can think of the space as a box. You can only put integer values into this box. In C++ the integer type is written `int`.

You can declare an integer variable like this:

```
int [name of variable];
```

or with an initial value like this:

```
int [name of variable] = [initial value];
```

For instance:

```
int x;
int y = 1;
```

You can refer to a variable only after it's declared.

The name of a variable is an example of an identifier. A identifier is just a name. Identifiers in C++ can be made up of alphanumerics and the underscore. The first character of an identifier cannot be a digit. An identifier should not begin with an underscore. To make a variable readable, you can either capitalize the first character of each word in the variable's name or insert an underscore between words. For instance:

```
int numLives = 3;
int num_lives = 3;
```

Using an uninitialized integer variable will either cause an error or a random value is given. (This depends on your compiler.)

You can declare more than one variable in one statement with or without initialization:

```
int x = 0, y, z = 5000;
```

An expression can contain variables:

```
x + y + 2 * z - a
```

Such an expression is evaluated by referring to the values of the relevant variables at the time of evaluation. C++ evaluates according to the usual precedence and associative rules.

You can assign a variable the value of an expression. For instance

```
i = x + y + 2 * z - a;
```

An assignment statement like the above means:

1. Evaluate the expression on the right
2. Give the value on the right to the variable on the left.

Printing a variable results in printing the value of the variable. Printing an expression results in printing the value of the expression after the expression is evaluated.

You can give an integer value to an integer variable by entering an integer at the keyboard:

```
int x;
std::cin >> x;
```

You can have an input statement for two variables. For instance:

```
int x, y;
std::cin >> x >> y;
```

In this case the integer values entered by the user from the keyboard can be separated any amount of whitespace.

On a 32-bit machine, a C++ `int` value ranges from -2^{31} to $2^{31} - 1$, i.e.

-2,147,483,648 to 2,147,483,647

Note: At this point you cannot declare a variable twice, i.e. the following is an error:

```
int x = 0;
int x = 42;
```

Later when we talk about scopes, I'll show it can be done, if variables (with the same name) are declared in different scopes.

Exercises.

1. Write a program that does the following when you execute it:

```
Enter an integer: 135
The rightmost digit is 5.
```

Here's another execution of the same program:

```
Enter an integer: 42
The rightmost digit is 2.
```

2. According to Albert Einstein, $E = mc^2$. Write a program that prompts the user for an integer value for m, an integer value for c, and prints the value of E according to the formula.

3. According to Elbert Ainstein, your IQ is given by the following formula:

IQ = 3 * w / h + (3 + f) / 42

where w is your waist (in inches), h is your height (in inches) and f is the number of fingers on your hands. The divisions are all integer divisions (i.e. quotients). Write a program that prompts the user for w, h, and f and print his/her IQ. (No ... the software won't sell.)

4. The following is a very imprtant formula that appears frequently in math and computer science:

$$s = \frac{n(n-1)}{2}$$

It's the sum of integer from 1 to n; it's also the number of ways to choose two out of a total of n toppings when you're at an ice cream shop. First work out the value of s for n = 1, 5, and 10 by hand. Write a program that prompts the user for a value of n and prints the value of s using the above formula. Test your program against the values you computed earlier.

5. What's wrong with this program?

```
#include <iostream>

int main()
{
    std::cout << "rectangle area!!!" << std::endl;
    std::cout << "length: ";
    std::cin >> w;
    std::cout << "width: ";
    std::cin >> h;
    std::cout << "area = " << wh << std::cout;

    return 0;
}
```

Now verify your correction with C++.

6. Write down the output of this program or explain why it won't run:

```
#include <iostream>

int main()
{
    int x = 1;
    int y = 2;
    int z = x + y;
    std::cout << x << y << z << std::endl;
    x = x + z;
    std::cout << y << x << z << std::endl;
    y = x * z;
    std::cout << z << x << y << std::endl;

    return 0;
}
```

Verify with your C++ compiler.

7. Write down the output of this program or explain why it won't run:

```
#include <iostream>

int main()
{
    int age;
    std::cin >> age;
    std::cout << "age:" << age << std::endl;

    int age = age + 1;
    std::cout << "next year:" << age << std::endl;

    return 0;
}
```

Verify with your C++ compiler.

8. Write down the output of this program or explain why it won't run:

```
#include <iostream>

int main()
{
    int 1st_prize = 300000;
    int 2nd_prize = 200000;
    int 3rd_prize = 100000;

    std::cout << 1st_prize << std::endl
              << 2nd_prize << std::endl
              << 3rd_prize << std::endl

    return 0;
}
```

Verify with your C++ compiler.

9. Write down the output of this program or explain why it won't run:

```
#include <iostream>

int main()
{
    int a = 0, b = 1;
    std::cout << a << ' ' << b << std::endl;

    int t = a;
    b = a;
    a = t;
    std::cout << a << ' ' << b << std::endl;

    return 0;
}
```

Verify with your compiler.

10. What is the output of this program?

```
#include <iostream>

int main()
{
    int i = 0, s = 0;
    std::cout << i << ' ' << s << '\n';

    s = s + i * i * i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i * i * i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i * i * i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    s = s + i * i * i;
    i = i + 1;
    std::cout << i << ' ' << s << '\n';

    return 0;
}
```

11. We have been initializing variables with constant integer values. What if we initialize an integer variable with the value of an expression? Can we do that? Circle one: YES NO. Now run this program where we try to initialize `k` with the value of an expression:

```
#include <iostream>

int main()
{
    int i = 0, j = 1;
    int k = i * j + j + 5;

    return 0;
}
```

12. What is the output of this program?

```
#include <iostream>

int main()
{
    int f = 0, g = 1, h = f + g;
    std::cout << f << ' ' << g << ' ' << h << '\n';

    f = g;
    g = h;
    h = f + g;
    std::cout << f << ' ' << g << ' ' << h << '\n';

    f = g;
    g = h;
    h = f + g;
    std::cout << f << ' ' << g << ' ' << h << '\n';

    f = g;
    g = h;
    h = f + g;
    std::cout << f << ' ' << g << ' ' << h << '\n';

    f = g;
    g = h;
    h = f + g;
    std::cout << f << ' ' << g << ' ' << h << '\n';

    return 0;
}
```

13. Here's some math: The derivative of x^n is nx^{n-1}. For instance the derivative of x^3 is $3x^2$. You don't have to understand the math behind it or what ithis “derivative thingy” is good for. You just need to write a program that prompts for n and then prints the derivative. Here's an execution of the program where the user enters 3:

```
n = 3

 d  3     2
-- x  = 3x
dx
```

and if the user enters 7 for n you get:

```
n = 7

 d  7     6
-- x  = 7x
dx
```

And if the user enters 1324 you get

```
n = 1324

 d  1324        1323
-- x     = 1324x
dx
```

14. What's wrong with this program?

```
#include <iostream>

int main()
{
    std::cout << "triangle area calculator"
              << std::endl;
    int base;
    std::cout << "base: ";
    std::cin >> base;
    std::cout << "height: ";
    std::cin >> height;
    std::cout << "area = "
              << 1/2 * base * height
              << std::cout;

    return 0;
}
```

Now verify your correction with C++.