5.3 — Modulus and Exponentiation

BY ALEX ON AUGUST 17TH, 2019 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 13TH, 2020

The modulus operator

The **modulus operator** (also informally known as the *remainder operator*) is an operator that returns the remainder after doing an integer division. For example, 7 / 4 = 1 remainder 3. Therefore, 7 % 4 = 3. As another example, 25 / 7 = 3 remainder 4, thus 25 % 7 = 4. Modulus only works with integer operands.

Modulus is most useful for testing whether a number is evenly divisible by another number: if *x % y* evaluates to 0, then we know that *x* is evenly divisible by *y*.

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int x{};
    std::cin >> x;

    std::cout << "Enter another integer: ";
    int y{};
    std::cin >> y;

    std::cout << "The remainder is: " << x % y << '\n';

    if ((x % y) == 0)
        std::cout << x << " is evenly divisible by " << y << '\n';
    else
        std::cout << x << " is not evenly divisible by " << y << '\n';

    return 0;
}
```

Here are a couple runs of this program:

```
Enter an integer: 6
Enter another integer: 3
The remainder is: 0
6 is evenly divisible by 3


Enter an integer: 6
Enter another integer: 4
The remainder is: 2
6 is not evenly divisible by 4
```

Modulus with negative numbers

The modulus operator can also work with negative operands. As of C++11, x % y always returns results with the sign of *x*.

Running the above program:

```
Enter an integer: -6
Enter another integer: 4
The remainder is: -2
-6 is not evenly divisible by 4


Enter an integer: 6
Enter another integer: -4
The remainder is: 2
6 is not evenly divisible by -4
```

In both cases, you can see the remainder takes the sign of the first operand.

Warning

Prior to C++11, modulus with a negative operand could result in either a positive or negative result. This was made predictable in C++11.

Where's the exponent operator?

You'll note that the ^ operator (commonly used to denote exponentiation in mathematics) is a *Bitwise XOR* operation in C++ (covered in lesson **O.3 -- Bit manipulation with bitwise operators and bit masks**). C++ does not include an exponent operator.

To do exponents in C++, #include the <cmath> header, and use the pow() function:

```
#include <cmath>

double x{ std::pow(3.0, 4.0) }; // 3 to the 4th power
```

Note that the parameters (and return value) of function pow() are of type double. Due to rounding errors in floating point numbers, the results of pow() may not be precise (even if you pass it integers or whole numbers).

If you want to do integer exponentiation, you're best off using your own function to do so. The following function implements integer exponentiation (using the non-intuitive "exponentiation by squaring" algorithm for efficiency):

```
#include <cstdint> // for std::int_fast64_t

// note: exp must be non-negative
std::int_fast64_t pow(int base, int exp)
{
    std::int_fast64_t result{ 1 };
    while (exp)
    {
        if (exp & 1)
            result *= base;
        exp >>= 1;
        base *= base;
    }

    return result;
}
```

Don't worry if you don't understand how this function works -- you don't need to understand it in order to call it.

```
#include <iostream>
```

```
#include <cstdint> // for std::int_fast64_t

// note: exp must be non-negative
std::int_fast64_t powint(int base, int exp)
{
    std::int_fast64_t result{ 1 };
    while (exp)
    {
        if (exp & 1)
            result *= base;
        exp >>= 1;
        base *= base;
    }

    return result;
}

int main()
{
    std::cout << powint(7, 12); // 7 to the 12th power

    return 0;
}
```

Produces:

```
13841287201
```

Warning

In the vast majority of cases, integer exponentiation will overflow the integral type. This is likely why such a function wasn't included in the standard library in the first place.

Quiz time

Question #1

What does the following expression evaluate to? `6 + 5 * 4 % 3`

Show Solution

Question #2

Write a program that asks the user to input an integer, and tells the user whether the number is even or odd. Write a function called isEven() that returns true if an integer passed to it is even, and false otherwise. Use the modulus operator to test whether the integer parameter is even.

Hint: You'll need to use if statements and the comparison operator (==) for this program. See lesson **4.9 -- Boolean values** if you need a refresher on how to do this.

Your program should match the following output:

```
Enter an integer: 5
5 is odd
```

Show Solution

5.4 -- Increment/decrement operators, and side effects

Index

5.2 -- Arithmetic operators

C++ TUTORIAL | PRINT THIS POST

30 comments to 5.3 — Modulus and Exponentiation

Tom
February 9, 2020 at 9:42 pm · Reply

I'm very confused by what's happening here:

```
1 if (isEven(x))
```

We have an if statement that runs a function and gets back either a 1 or 0, but we don't ask the if statement to check if it's a 1 or 0, we just accept whatever we get and move on to the next line, and if we got something different than we do the else. Am I mistaken or are we just neglecting to check what the returned value is for the first if block and just finding out at runtime and assigning the correct cout sentence to it and calling it a day? Why not say "if (isEven(x) = 0)"? Is it because there can only be 2 possible responses and we just don't care about specifying? Very confused.

RJ
February 10, 2020 at 4:44 pm · Reply

The syntax of an if statement is, if (condition) statement; the condition will always evaluate to a true or false value that's the purpose of and if statement. Your not ignoring the return value from isEven(x), basicly if "isEven(x)" returns 1 or any other number that's not zero (true) to the compiler it looks like this

```
1 if (true)
```

then the statement specified will execute, and if it returned 0 (false) it would look like this to the compiler

```
1 if (false)
```

.

Chris
February 2, 2020 at 5:50 pm · Reply

Ged,
To avoid using std:: before cout and pow, try using:

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
```

Papi
February 2, 2020 at 3:19 pm · Reply

Hello.

```
1  #include <iostream>
2
3  int readNumber()
4  {
5      std::cout << "Enter an integer: ";
6      int integer{ 0 };
7      std::cin >> integer;
8      return integer;
9  }
10
11 void isEven(int number)
12 {
13     if (number % 2 == 0)
14         std::cout << "Number is even.";
15     else
16         std::cout << "Number is odd.";
17 }
18
19 int main()
20 {
21     int number{ readNumber() };
22     isEven(number);
23     return 0;
24 }
```

How bad is this code? Ignore the formatting pls

Raton
February 3, 2020 at 10:13 am · Reply

Hi! I think it's recommended not to put zero at line 6 because the value of "integer" is immediately discarded (so the 0 doesn't really mean anything, just put: int integer{};). You

can also add parenthesis at line 13: if ((number % 2) == 0), to make it easier to read. If you really want to follow best practices you could maybe also make isEven() return a Boolean value (the number's parity) and put the output part (with std::cout, etc.) in main(), or in another function (called in main()). You can also add a line feed (\n) at the end of the strings in line 14 and 16 to separate your output from the things printed by the compiler.

Tony98
January 26, 2020 at 9:58 am · Reply

I'm always completing the exercises in different ways. Is it ok if I always create multiple simple functions to do all the jobs? Like this for this specific exercise:

```
1  bool isEven(int userNumber)
2  {
3      if (userNumber % 2 == 0)
4          return true;
5  }
6
7
8  int getUserValue()
9  {
10     std::cout << "Enter an int: ";
11     int x{};
12     std::cin >> x;
13
14     return x;
15 }
16
17 void printResult(int userValue)
18 {
19     if (isEven(userValue))
20         std::cout << userValue << " is even";
21     else
22         std::cout << userValue << " is odd";
23
24 }
25
26 int main()
27 {
28     int x{ getUserValue() };
29     isEven(x);
30     printResult(x);
31
32     return 0;
33 }
```

I find it much easier to create first all the simple functions and then call them on main, what do you think? Thanks again for the reply!

nascardriver
January 27, 2020 at 3:44 am · Reply

Yes, it's better that way. Some of the examples are so short that the solution just crammed everything into 1 or 2 functions.
Your call to `isEven` in line 29 doesn't do anything.

giang

January 13, 2020 at 12:02 am · Reply

So can we use the pow() function for integers?? Because I try this function for integers and it worked

nascardriver
January 13, 2020 at 2:11 am · Reply

`std::pow` only operates on floating point numbers.

```
1 int i{ std::pow(1, 2) }; // error
```

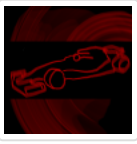
Kyle
January 3, 2020 at 6:25 pm · Reply

I did it different than you but ended up with the same results, is this fine? I am also a lot more comfortable with the way I did when compared to the way you did it, is that bad and should I try to change that mindset? I would also like to thank you for this book as I'm going to call it, it has been a God send.

```
1  #include <iostream>
2
3  int getInput()
4  {
5      std::cout << "Enter an integer: ";
6      int a{};
7      std::cin >> a;
8
9      return a;
10 }
11
12 void isEven(int a)
13 {
14     if ((a % 2) == 0)
15         std::cout << a << " is even.";
16     else
17         std::cout << a << " is odd.";
18 }
19
20 int main()
21 {
22     int a{ getInput() };
23     isEven(a);
24
25     return 0;
26 }
```

Kyle
January 5, 2020 at 4:22 am · Reply

I was examining question 2 again and I just realized you requested that we make a function that returns true or false, I missed that. Never mind my bit of code.



nascardriver
January 5, 2020 at 6:24 am · Reply

Your code still shows that you understand the concepts asked for in this quiz. If your program prints anything, it's good to print a line feed (\n) at the end to prevent mixing

output with other programs.

Kyle
January 5, 2020 at 7:14 am · Reply

Thank you so much for your feedback. I normally do put '\n' at the end of my prints but I often forget to, this was one of those cases.

chai
December 5, 2019 at 8:23 am · Reply

```
bool isOdd(int x)
{
    return (x % 2);
}
```

A little bit shorter but just as good?

nascardriver
December 6, 2019 at 2:18 am · Reply

You're converting from int to bool. It does what it should, but can be harder to read than an explicit conversion via `==`.

koe
December 19, 2019 at 10:40 pm · Reply

I had the same idea.

```
bool is_even(int test_val)
{
    return !(test_val % 2);
}
```

Would this code be slower than the quiz solution, or is it just a matter of clarity?

nascardriver
December 20, 2019 at 1:15 am · Reply

It will probably produce exactly the same binary as the quiz. Your compiler is great at optimizing simple things. Do what you find easier to read, I prefer the comparison to 0.

Raul
December 5, 2019 at 5:16 am · Reply

Thanks for the well-structured material!
My approach:

```
#include <iostream>

bool isEven(int x)
```

```
{
    if (x % 2 == 0)
        std::cout << x << " is even.\n";
    else
        std::cout << x << " is odd.\n";
    return false;
}

int getValue()
{
    std::cout << "Enter an integer : ";
    int x{};
    std::cin >> x;

    return x;
}

int main()
{
    int valueStored { getValue() };
    isEven(valueStored);

    return 0;
}
```

nascardriver
December 5, 2019 at 5:19 am · Reply

hi Raul!

`isEven` isn't supposed to print anything, but return false/true depending on whether or not `x` is even. Try to implement it in a way that `main` does the printing.

Raul
December 5, 2019 at 6:20 am · Reply

Thanks for the intervention, nascardriver.

```
#include <iostream>

bool isEven(int x)
{
    if (x % 2 == 0)
        return true;
    else
        return false;
}

int getValue()
{
    std::cout << "Enter an integer : ";
    int x{};
    std::cin >> x;

    return x;
}

int main()
{
    int valueStored{ getValue() };
```

```
    isEven(valueStored);

    if(isEven(valueStored))
        std::cout << valueStored << " is even.\n";
    else
        std::cout << valueStored << " is odd.\n";

    return 0;
}
```

nascardriver
December 5, 2019 at 6:22 am · Reply

That's it, good job! `isEven` can be simplified even further, have a look at the solution for an explanation.

Alex3>
November 8, 2019 at 11:56 pm · Reply

This is my way of solving this task:

```
bool isTrue(int x)
{
    cout << "Enter an intiger: "; cin >> x;
    if (x % 2 < 1)
        cout << x << " is even";
    else
        cout << x << " is odd";
    return 0;

}

int main()
{
    int x{};
    isTrue(x);
    return 0;
}
```

nascardriver
November 9, 2019 at 5:22 am · Reply

Hi!

You're not using the return value of `isTrue`. Declare it `void`.
`std::cin >> x` should be on its own line for better readability.
If your program prints anything, the last thing it prints should be a line feed.
"isTrue" is a poor name, it doesn't describe what the function does.

Knight
November 8, 2019 at 10:09 pm · Reply

```
// Example program
#include <iostream>
#include <string>
```

```
4
5  bool isEven(int x)
6  {
7    if(x%2==0)
8    std::cout<<x<<" is even";
9    else
10   std::cout<<x<<" is odd";
11   return 0;
12 }
13
14
15 int main()
16 {
17   std::cout<<"Enter an integer: ";
18   int x{0};
19   std::cin>>x;
20   std::cout<<isEven(x);
21
22 }
```

How to fix this code? 0 appeared on the result. Thanks.

nascardriver
November 9, 2019 at 5:19 am · Reply

`isEven` always returns `false` (Line 11). It appears you don't want to use the return value at all. Declare `isEven` `void` and don't print in line 20.
Return values are covered in lesson 2.2.

Knight
November 10, 2019 at 1:08 am · Reply

Thanks, Nascardriver. I should read carefully and slowly.

Ged
October 27, 2019 at 6:29 am · Reply

Why do we need to write std:: for pow? Cause it allows us to use it without std. Both codes work.

```
1 #include <iostream>
2 #include <cmath>
3
4 int main()
5 {
6     std::cout << std::pow(2.0,3.0);
7     return 0;
8 }
```

```
1 #include <iostream>
2 #include <cmath>
3
4 int main()
5 {
6     std::cout << pow(2.0,3.0);
7     return 0;
8 }
```

nascardriver
October 27, 2019 at 6:37 am · Reply

declares its contents inside the `std` namespace. It also declares them outside for C compatibility, so you can use them without `std::`. To prevent name collisions and to make sure your code still works if the outside-declaration should ever be removed, I recommend using `std::pow`.

Hadal
August 28, 2019 at 11:21 am · Reply

You forgot to include the word 'evenly' in the code output examples.

Alex
September 1, 2019 at 1:37 pm · Reply

Fixed. Thanks!