O.4 — Converting between binary and decimal

BY ALEX ON JUNE 17TH, 2007 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 16TH, 2020

Consider a normal decimal number, such as 5623. We intuitively understand that these digits mean (5 * 1000) + (6 * 100) + (2 * 10) + (3 * 1). Because there are 10 decimal numbers, the value of each subsequent digit to the left increases by a factor of 10.

Binary numbers work the same way, except because there are only 2 binary digits (0 and 1), the value of each digit increases by a factor of 2. Just like commas are often used to make a large decimal number easy to read (e.g. 1,427,435), we often write binary numbers in groups of 4 bits to make them easier to read (e.g. 1101 0101).

The following table counts to 15 in decimal and binary:

Decimal Value	Binary Value
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Converting binary to decimal

In the following examples, we assume that we're dealing with unsigned integers.

Consider the 8 bit (1 byte) binary number 0101 1110. Binary 0101 1110 means (0 * 128) + (1 * 64) + (0 * 32) + (1 * 16) + (1 * 8) + (1 * 4) + (1 * 2) + (0 * 1). If we sum up all of these parts, we get the decimal number 64 + 16 + 8 + 4 + 2 = 94.

Here is the same process in table format. We multiply each binary digit by its digit value (determined by its position). Summing up all these values gives us the total.

Converting 0101 1110 to decimal:

Binary digit	0	1	0	1	1	1	1	0
* Digit value	128	64	32	16	8	4	2	1
= Total (94)	0	64	0	16	8	4	2	0

Let's convert 1001 0111 to decimal:

Binary digit	1	0	0	1	0	1	1	1
* Digit value	128	64	32	16	8	4	2	1
= Total (151)	128	0	0	16	0	4	2	1

1001 0111 binary = 151 in decimal.

This can easily be extended to 16 or 32 bit binary numbers simply by adding more columns. Note that it's easiest to start on the right end, and work your way left, multiplying the digit value by 2 as you go.

Method 1 for converting decimal to binary

Converting from decimal to binary is a little more tricky, but still pretty straightforward. There are two good methods to do this.

The first method involves continually dividing by 2, and writing down the remainders. The binary number is constructed at the end from the remainders, from the bottom up.

Converting 148 from decimal to binary (using r to denote a remainder):

148 / 2 = 74 r0
74 / 2 = 37 r0
37 / 2 = 18 r1
18 / 2 = 9 r0
9 / 2 = 4 r1
4 / 2 = 2 r0
2 / 2 = 1 r0
1 / 2 = 0 r1

Writing all of the remainders from the bottom up: 1001 0100

148 decimal = 1001 0100 binary.

You can verify this answer by converting the binary back to decimal:

(1 * 128) + (0 * 64) + (0 * 32) + (1 * 16) + (0 * 8) + (1 * 4) + (0 * 2) + (0 * 1) = 148

Method 2 for converting decimal to binary

The second method involves working backwards to figure out what each of the bits must be. This method can be easier with small binary numbers.

Consider the decimal number 148 again. What's the largest power of 2 that's smaller than 148? 128, so we'll start there.

Is 148 >= 128? Yes, so the 128 bit must be 1. 148 - 128 = 20, which means we need to find bits worth 20 more.
Is 20 >= 64? No, so the 64 bit must be 0.

Is 20 >= 32? No, so the 32 bit must be 0.
Is 20 >= 16? Yes, so the 16 bit must be 1. 20 - 16 = 4, which means we need to find bits worth 4 more.

Is 4 >= 8? No, so the 8 bit must be 0.
Is 4 >= 4? Yes, so the 4 bit must be 1. 4 - 4 = 0, which means all the rest of the bits must be 0.

148 = (1 * 128) + (0 * 64) + (0 * 32) + (1 * 16) + (0 * 8) + (1 * 4) + (0 * 2) + (0 * 1) = 1001 0100

In table format:

Binary number	1	0	0	1	0	1	0	0
* Digit value	128	64	32	16	8	4	2	1
= Total (148)	128	0	0	16	0	4	0	0

Another example

Let's convert 117 to binary using method 1:

117 / 2 = 58 r1
58 / 2 = 29 r0
29 / 2 = 14 r1
14 / 2 = 7 r0
7 / 2 = 3 r1
3 / 2 = 1 r1
1 / 2 = 0 r1

Constructing the number from the remainders from the bottom up, 117 = 111 0101 binary

And using method 2:

The largest power of 2 less than 117 is 64.

Is 117 >= 64? Yes, so the 64 bit must be 1. 117 - 64 = 53.
Is 53 >= 32? Yes, so the 32 bit must be 1. 53 - 32 = 21.
Is 21 >= 16? Yes, so the 16 bit must be 1. 21 - 16 = 5.

Is 5 >= 8? No, so the 8 bit must be 0.
Is 5 >= 4? Yes, so the 4 bit must be 1. 5 - 4 = 1.
Is 1 >= 2? No, so the 2 bit must be 0.
Is 1 >= 1? Yes, so the 1 bit must be 1.

117 decimal = 111 0101 binary.

Adding in binary

In some cases (we'll see one in just a moment), it's useful to be able to add two binary numbers. Adding binary numbers is surprisingly easy (maybe even easier than adding decimal numbers), although it may seem odd at first because you're not used to it.

Consider two small binary numbers:
0110 (6 in decimal) +
0111 (7 in decimal)

Let's add these. First, line them up, as we have above. Then, starting from the right and working left, we add each column of digits, just like we do in a decimal number. However, because a binary digit can only be a 0 or a 1, there

are only 4 possibilities:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0, carry a 1 over to the next column

Let's do the first column:

```
0110 (6 in decimal) +
0111 (7 in decimal)
----
   1
```

0 + 1 = 1. Easy.

Second column:

```
 1
0110 (6 in decimal) +
0111 (7 in decimal)
----
  01
```

1 + 1 = 0, with a carried one into the next column

Third column:

```
11
0110 (6 in decimal) +
0111 (7 in decimal)
----
 101
```

This one is a little trickier. Normally, 1 + 1 = 0, with a carried one into the next column. However, we already have a 1 carried from the previous column, so we need to add 1. Thus, we end up with a 1 in this column, with a 1 carried over to the next column

Last column:

```
11
0110 (6 in decimal) +
0111 (7 in decimal)
----
1101
```

0 + 0 = 0, but there's a carried 1, so we add 1. 1101 = 13 in decimal.

Now, how do we add 1 to any given binary number (such as 1011 0011)? The same as above, only the bottom number is binary 1.

```
       1  (carry column)
1011 0011 (original binary number)
0000 0001 (1 in binary)
```

```
---------
1011 0100
```

Signed numbers and two's complement

In the above examples, we've dealt solely with unsigned integers. In this section, we'll take a look at how signed numbers (which can be negative) are dealt with.

Signed integers are typically stored using a method known as **two's complement**. In two's complement, the leftmost (most significant) bit is used as the sign bit. A 0 sign bit means the number is positive, and a 1 sign bit means the number is negative.

Positive signed numbers are represented in binary just like positive unsigned numbers (with the sign bit set to 0).

Negative signed numbers are represented in binary as the bitwise inverse of the positive number, plus 1.

Converting integers to binary two's complement

For example, here's how we represent -5 in binary two's complement:

First we figure out the binary representation for 5: 0000 0101
Then we invert all of the bits: 1111 1010
Then we add 1: 1111 1011

Converting -76 to binary:

Positive 76 in binary: 0100 1100
Invert all the bits: 1011 0011
Add 1: 1011 0100

Why do we add 1? Consider the number 0. If a negative value was simply represented as the inverse of the positive number, 0 would have two representations: 0000 0000 (positive zero) and 1111 1111 (negative zero). By adding 1, 1111 1111 intentionally overflows and becomes 0000 0000. This prevents 0 from having two representations, and simplifies some of the internal logic needed to do arithmetic with negative numbers.

Converting binary two's complement to integers

To convert a two's complement binary number back into decimal, first look at the sign bit.

If the sign bit is 0, just convert the number as shown for unsigned numbers above.

If the sign bit is 1, then we invert the bits, add 1, then convert to decimal, then make that decimal number negative (because the sign bit was originally negative).

For example, to convert 1001 1110 from two's complement into a decimal number:
Given: 1001 1110
Invert the bits: 0110 0001
Add 1: 0110 0010
Convert to decimal: (0 * 128) + (1 * 64) + (1 * 32) + (0 * 16) + (0 * 8) + (0 * 4) + (1 * 2) + (0 * 1) = 64 + 32 + 2 = 98
Since the original sign bit was negative, the final value is -98.

Why types matter

Consider the binary value 1011 0100. What value does this represent? You'd probably say 180, and if this were a standard unsigned binary number, you'd be right.

However, if this value was stored using two's complement, it would be -76.

And if the value were encoded some other way, it could be something else entirely.

So how does C++ know whether to print a variable containing binary 1011 0100 as 180 or -76?

In case the section title didn't give it away, this is where types come into play. The type of the variable determines both how a variable's value is encoded into binary, and decoded back into a value. So if the variable type was an unsigned integer, it would know that 1011 0100 was standard binary, and should be printed as 180. If the variable was a signed integer, it would know that 1011 0100 was encoded using two's complement (now guaranteed as of C++20), and should be printed as -76.

What about converting floating point numbers from/to binary?

How floating point numbers get converted from/to binary is quite a bit more complicated, and not something you're likely to ever need to know. However, if you're curious, see **this site**, which does a good job of explaining the topic in detail.

Quiz time

Question #1

Convert 0100 1101 to decimal.

Show Solution

Question #2

Convert 93 to an 8-bit unsigned binary number. Use both methods above.

Show Solution

Question #3

Convert -93 to an 8-bit signed binary number (using two's complement).

Show Solution

Question #4

Convert 1010 0010 to an unsigned decimal number.

Show Solution

Question #5

Convert 1010 0010 to a signed decimal number (assume two's complement).

Show Solution

Question #6

Write a program that asks the user to input a number between 0 and 255. Print this number as an 8-bit binary number (of the form #### ####). Don't use any bitwise operators. Don't use `std::bitset`.

Show Hint

Show Hint

Show Solution

6.1 -- Compound statements (blocks)

Index

O.3 -- Bit manipulation with bitwise operators and bit masks

C++ TUTORIAL | PRINT THIS POST

437 comments to O.4 — Converting between binary and decimal

« Older Comments 1 ... 5 6 7

Sam
January 15, 2020 at 8:53 pm · Reply

Question #5 solution has a grammatical error.

"First, start with out binary number: 1010 0010"

Change "out" to "our"! :]

nascardriver
January 16, 2020 at 2:30 am · Reply

Lessons amended, thanks!

Suyash
January 15, 2020 at 4:53 pm · Reply

The final question is a standard one (for anyone who has taken a programming course in college)... Usually, the standard solution is to utilize the first method of continuously dividing by 2 and printing the remainder...

And, while due to the fact, that the course has yet to introduce loops (as well as conditionals with multiple statements), I decided to implement the above approach using recursion but it made printing the binary number in the required format difficult...

Hence, I decided to use the second approach of comparing the powers of 2 with the decimal value and then, subtracting it (if necessary)... I decided to only use the concepts that have been discussed so far (in the tutorial series) to make it easier for other learners... I didn't want to confuse them by using constructs unfamiliar to them...

And, finally, keep up the great work!!

Here's my solution:

```
#include <iostream>
#include <cmath>    // For pow()

unsigned int getBitAt(unsigned int bit_pos, unsigned int decimal)
{
    // Use copy initialization for narrowing conversion from `double` to `unsigned int`.
    unsigned int bit_value = std::pow(2.0, bit_pos);

    // Get the value of bit at `bit_pos`.
    bool bit{ (decimal >= bit_value) ? true : false };

    // Only perform subtraction if either `decimal` is greater than `2 ^ bit_pos`
    // or `bit_pos` is not 0.
    if (bit && bit_pos) decimal -= bit_value;

    std::cout << bit;
    return decimal;
}

void printBinary(unsigned int decimal)
{
    std::cout << "Binary Form of " << decimal << " is ";

    decimal = getBitAt(7, decimal);
    decimal = getBitAt(6, decimal);
    decimal = getBitAt(5, decimal);
    decimal = getBitAt(4, decimal);

    std::cout << ' ';

    decimal = getBitAt(3, decimal);
```

```
    decimal = getBitAt(2, decimal);
    decimal = getBitAt(1, decimal);
    decimal = getBitAt(0, decimal);

    std::cout << '\n';
}

int main()
{
    std::cout << "Enter an integer between 0 and 255: ";
    unsigned int decimal{};
    std::cin >> decimal;

    printBinary(decimal);

    return 0;
}
```

yoozer_0
January 14, 2020 at 12:36 am · Reply

Did it like this before reading hints lol.

```
#include <iostream>


std::uint_fast16_t print_binary(std::uint_fast16_t x)
{
    std::uint_fast16_t one, two, three, four, five, six, seven, eight;

    one = x % 2;
    two = (x / 2) % 2;
    three = ((x / 2) / 2) % 2;
    four = (((x / 2) / 2) / 2) % 2;
    five = ((((x / 2) / 2) / 2) / 2) % 2;
    six = (((((x / 2) / 2) / 2) / 2) / 2) % 2;
    seven = ((((((x / 2) / 2) / 2) / 2) / 2) / 2) % 2;
    eight = (((((((x / 2) / 2) / 2) / 2) / 2) / 2) / 2) % 2;

    std::cout << x << " = " << eight << seven << six << five << " " << four << three << two

    return 0;

}

int main()
{
    std::cout << "Enter a number between 0 and 255: ";
    std::uint_fast16_t x{};
    std::cin >> x;

    print_binary(x);

    return 0;
}
```

koe
December 20, 2019 at 1:28 pm · Reply

Hi, for the quiz I had the same solution but the problem is previous lessons did not introduce conditionals with multiple statements. Only:

```
if (expression)
statement;
```

So I think to be compliant with lessons the answer should be

```
#include <iostream>

//print binary digit 1 and subtract the power of 2 from val
int output_one(int val, int pow_of_two)
{
    std::cout << 1;
    return val - pow_of_two;
}


//val is decimal number
//pow_of_two is a binary digit in decimal form, to check if it's contained in val
int check_and_print(int val, int pow_of_two)
{
    //is the power of two contained within val? print binary digit 1 and return the remaind
    if (val >= pow_of_two)
        return output_one(val, pow_of_two);

    //it's not, print binary digit 0 and don't change val
    std::cout << 0;
    return val;
}


int main()
{
    //get user input
    std::cout << "Enter an integer from 0 to 255: ";
    int user_int{};
    std::cin >> user_int;

    //print in binary form by extracting powers of two
    user_int = check_and_print(user_int, 128);
    user_int = check_and_print(user_int, 64);
    user_int = check_and_print(user_int, 32);
    user_int = check_and_print(user_int, 16);
    std::cout << " ";
    user_int = check_and_print(user_int, 8);
    user_int = check_and_print(user_int, 4);
    user_int = check_and_print(user_int, 2);
    user_int = check_and_print(user_int, 1);
    std::cout << '\n';

    return 0;
}
```

nascardriver

December 22, 2019 at 4:34 am · Reply

It's surprising no one else noticed it, but I couldn't find a multi-statement if in a previous lesson. Lesson updated, thanks!

Ikenna
December 9, 2019 at 9:05 am · Reply

Hey there,
Just made a program in response to the last quiz. No bitwise operators, no bitsets. If anyone has any suggestions or ideas for improving the code that will appreciated. Thank you.

Main.cpp

```
// BitsPlay.cpp : This file contains the 'main' function. Program execution begins and ends
/*
Goal -- Write a program that asks the user to input a number between 0 and 255.
Print this number as an 8-bit binary number (of the form #### ####).
Don't use any bitwise operators. Don't use std::bitset.
*/

#include <iostream>
#include <vector>
#include "InputFormattingFunctions.h"


int main()
{

    std::cout << "Enter an integer. (It may only be between 0 and 255) :" << '\n';

    int input {};
    std::cin >> input;

    validateInput(input);
    std::vector<bool> bits { generateBitVector(input) };
    bits = reverseBitPositions(bits);

    std::cout << "The binary form of " << input << " is ";
    formatAndPrintBits(bits);

    std::cout << '\n';

}
```

Functions.h

```
#ifndef IFF_H
#define IFF_H

#include <iostream>
#include <vector>

void validateInput(int input);

//Creates a vector object populated with bits formed from modulus arithmetic.
std::vector<bool> generateBitVector(int integer);

//The name explains it. Reverses the position of each bit from ascending index order to des
std::vector<bool> reverseBitPositions(std::vector<bool> bits);

//Simply adds necessary amount of preceding 0's and adds space per 4 bits giving it this fo
void formatAndPrintBits(std::vector<bool> bits);

#endif
```

Functions.cpp

```cpp
#include <iostream>
#include <vector>

void validateInput(int input) {
    bool isValid{ (input >= 0) && (input <= 255) };

    //Continually reject all integers out of the range
    while (!isValid) {
        std::cout << "The number is out of the expected range. Please try again: " << "\n";
        std::cin >> input;
        isValid = (input >= 0) && (input <= 255);
    }
}

std::vector<bool> generateBitVector(int integer) {

    bool bit{};                         //The 1's and 0's that will get printed to the conso
    std::vector<bool> bitVector{};      //Holds each bit produced. Needed for later.

    /*
    First Zero Case
    Recognizes the fact that a division of 0 and a remainder of 1 contributes a bit.
    This is crucial since a division of 0 and a remainder of 0 adds infinite 0 bits and hen
    We don't want that.
    */
    bool firstZeroCase{ integer == 0 && bit != 0 };


    while (integer > 0 || firstZeroCase) {

        //Modulus arithmetic: Divide each integer by 2 and store the remainder.
        bit = integer % 2;

        integer = integer / 2;

        //Add each bit to the vector.
        bitVector.push_back(bit);


    }
    return bitVector;

}

std::vector<bool> reverseBitPositions(std::vector<bool> bits) {

    std::vector<bool> formattedBits{};    //Holds the bits in reverse order.
    unsigned int counter { bits.size() - 1 };

    while (counter >= 0) {

        //spaceCharTracker += 1;
        formattedBits.push_back(bits[counter]);

        //Handles the assertion error that would have been created using a for loop with un
        if (counter == 0) {
            break;
        }
        --counter;
    }
```

```
    return formattedBits;
}

void formatAndPrintBits(std::vector<bool> bits) {
    int spaceCharTracker{ 0 };   //Tracks the sequence in which space characters are added

    unsigned int numOfPrecedingZeros{ 8 - bits.size() }; //Tracks the number of 0's coming

    //Insert only as much 0's as the representation needs.
    while (numOfPrecedingZeros > 0) {
        bits.insert(bits.begin(), 0);
        --numOfPrecedingZeros;
    }

    //Print each bit figure from the vector to the console.
    for (int i{ 0 }; i < bits.size(); ++i) {
        std::cout << bits[i];
        ++spaceCharTracker;

        //Add a space character for every 4 digits of bits printed to the console.
        if ((spaceCharTracker % 4) == 0) {
            std::cout << ' ';

        }
    }
}
```

Thanks!

Anderson
December 10, 2019 at 10:38 am · Reply

The program works indeed. Only problem is that it returns an assertion error when user inputs 0.
The cause of this is in the generateBitVector() function. Here's one way to fix that, Simply initialize the 'bit' variable to value of 1 and then put a conditional statement 'if' in this case to make the program breaks out of the loop when the input is down to 0:

Functions.cpp

```
//....
std::vector<bool> generateBitVector(int input) {
 bool bit { 1 };
 std::vector<bool> bitVector {};
 //....
 bool firstZeroCase{ integer == 0 && bit != 0 };


    while (integer > 0 || firstZeroCase) {

        //Modulus arithmetic: Divide each integer by 2 and store the remainder.
        bit = integer % 2;

        integer = integer / 2;

        //Add each bit to the vector.
        bitVector.push_back(bit);

        if (input == 0)
          break;
    }
```

```
22      return bitVector;
23  }
```

That makes it work with 0 as an input now. You're welcome.

Anderson
December 8, 2019 at 4:45 am · Reply

Hi!

I'm still so confused about which digit of a binary figure is the sign bit.
For example :

```
signed char myFlag { 0b1011 0010 } //assuming this is C++ 14 or later
```

Which is the sign bit in this case, '0' at index 0 or the '1' at index 7.
I'm sorry for asking this question but I need verification since I may have misunderstood the examples

nascardriver
December 9, 2019 at 3:34 am · Reply

You can't set the sign bit of literals, they'll be treated as an integer of higher width rather than setting the sign bit.
The sign bit is the leftmost bit. In your example, assuming the number was treated as an 8 bit number, the sign bit is 1.

Nick
December 7, 2019 at 5:23 pm · Reply

Hello! Thanks for the tutorials, very helpful!

Just wanted to point out that I implemented question 6 with static_cast<std::bitset<8>>(getIntFromUser()) -- perhaps to make it less ambiguous what the intent of the question is, could there be wording inserted to say "implement the method 2 algorithm from above to encode a decimal number to binary" or something to that effect? That would help make the intention clearer without having to look at hints, I think.

Again, thank you for all the effort you've put into this fantastic resource! Cheers!

nascardriver
December 8, 2019 at 5:49 am · Reply

Quiz updated to forbid the use of `std::bitset`. Thanks for pointing out the unclarity!

knight
December 2, 2019 at 9:14 pm · Reply

Hi, I am confused about the question 6 answer codes:

```
x = printandDecrementBit(x, 128);
   x = printandDecrementBit(x, 64);
   x = printandDecrementBit(x, 32);
   x = printandDecrementBit(x, 16);
```

Will C++ automatically save the previous counted data? Has lesson mentioned about this? Thanks.

nascardriver
December 3, 2019 at 4:38 am · Reply

`printandDecrementBit` returns the value of `x` or the new value of `x`, which we then store in `main`'s `x`. Return values are covered in lesson 2.2.

Blue
August 9, 2019 at 8:13 pm · Reply

Hello! Like everyone else I came up with a different solution to question 6. It works, but is it overly complicated and does it follow best practices? Thanks in advance.

functions.h

```
#ifndef FUCNTIONS_H
#define FUNCTIONS_H

int getInteger();
void printResult(int dec);

#endif
```

functions.cpp

```
#include <iostream>

bool isRangeValid(int dec)                // int decimal
{
    return dec >= 0 && dec <= 255;
}

int getInteger()
{
    std::cout << "Enter an integer between 0 and 255, inclusive: ";
    int dec{};
    std::cin >> dec;

    while (!isRangeValid(dec))
    {
        std::cout << "Invalid input! Try again: ";
        std::cin >> dec;
    }

    return dec;
}

int convertDecimalToBinary(int dec, int bin)        // int binary
{
    if (dec >= bin)
        return 1;

    return 0;
}

int decreaseDecimalValue(int dec, int bin)
{
    if (dec >= bin)
        return dec -= bin;

    return dec;
```

```
}

void printResult(int dec)
{
    int bin { 128 };

    while (bin >= 16)
    {
        std::cout << convertDecimalToBinary(dec, bin);
        dec = decreaseDecimalValue(dec, bin);
        bin /= 2;
    }
    std::cout << ' ';

    while (bin >= 1)
    {
        std::cout << convertDecimalToBinary(dec, bin);
        dec = decreaseDecimalValue(dec, bin);
        bin /= 2;
    }
    std::cout << '\n';
}
```

main.cpp

```
#include "functions.h"
#include <iostream>

int main()
{
    int dec { getInteger() };
    printResult(dec);

    return 0;
}
```

nascardriver
August 10, 2019 at 2:12 am · Reply

Hello!

- functions.cpp:34: You're not using `dec` after this line, don't modify `dec`. Use a regular `-`.
- functions.cpp:45-47, 53-55: Repetition. Move these lines into a function.

The rest looks good :0)

Blue
August 10, 2019 at 12:48 pm · Reply

Got it, will update accordingly. Enjoy the rest of your day!

Alexander S.
July 18, 2019 at 7:13 pm · Reply

There are no errors, but is this a valid solution to Q6 / are there general code errors I should be aware of? Thanks!

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

```
3
4   int powerCalc(int power) {
5       int times{ 0 };
6       while (power >= 1) {
7           power /= 2;
8           times++;
9       }
10
11      return times;
12  }
13
14  void decimalBinary(int value, int startPower) {
15      int currentPower{ startPower };
16      int currentValue{ value };
17
18
19      for (int i{ 0 }; i < powerCalc(startPower); i++) {
20          if (i % 4 == 0 && i != 0) {
21              std::cout << " ";
22
23          }
24
25
26          if (currentValue >= currentPower) {
27              std::cout << '1';
28              currentValue -= currentPower;
29              currentPower /= 2;
30          }    else {
31              std::cout << '0';
32              currentPower /= 2;
33
34          }
35
36      }
37
38  }
39
40
41  int main()
42  {
43      int valueS{};
44      int startPow{};
45      std::cout << "Please enter a number to convert to decimal\n";
46      std::cin >> valueS;
47      std::cout << "Please enter a power of 2 (Type '0' for list)\n";
48      std::cin >> startPow;
49      if (startPow == 0) {
50          std::cout << "Listing Powers of 2\n";
51          for (int i{ 0 }; i < 15; i++) {
52              std::cout << std::pow(2, i) << std::endl;
53          }
54          main();
55      }
56      decimalBinary(valueS, startPow);
57      return 0;
58  }
```

nascardriver
July 19, 2019 at 12:56 am · Reply

Using `main` causes undefined behavior (Line 54).

- Use ++prefix instead of postfix++ unless you need postfix++.
- Use single quotation marks for characters.
- Line 29, 32 are unconditional, move them out of the `if`.
- Use your editor's auto-formatting feature.
- You're calling `powerCalc` after every loop cycle, but it will return the same number every time.
- If your program prints anything, the last thing it prints should be a line feed.

Alexander S.
July 19, 2019 at 5:52 am · Reply

Is this better?

```
#include <iostream>
#include <cmath>
int main();
int powerCalc(int power) {
    int times{ 0 };
    while (power >= 1) {
        power /= 2;
        ++times;
    }
    return times;
}
void negativeBinary(int value) {
}
void decimalBinary(int value, int startPower) {
    int currentPower{ startPower };
    int currentValue{ value };
    int loopTimes{ powerCalc(startPower) };
    std::string binary{};
    if (value == 0) {
        std::cout << '0\n';
        main();
    } else if (value < 0) {
        negativeBinary(value);
    } else {
        for (int i{ 0 }; i < loopTimes; ++i) {
            if (i % 4 == 0 && i != 0) {
                binary += ' ';
            }
            if (currentValue >= currentPower) {
                binary += '1';
                currentValue -= currentPower;

            } else {
                binary += '0';
            }
            currentPower /= 2;
        }
    }
    std::cout << binary << std::endl;
}
int askQuestions() {
    int valueS{};
    int startPow{};
    std::string response{};
    std::cout << "Please enter an 8-bit integer to convert to decimal\n";
    std::cin >> valueS;
    if (valueS > 999999999) {
        std::cout << "Too Large of a Number";
```

```
49          return 0;
50      }
51      std::cout << "Please enter a power of 2 (Type '0' for list)\n";
52      std::cin >> startPow;
53      if (startPow == 0) {
54          std::cout << "Powers of 2 \n----------\n";
55          for (int i{ 0 }; i < 15; i++) {
56              std::cout << std::pow(2, i) << std::endl;
57          }
58          std::cout << "----------\n";
59          main();
60      }
61      decimalBinary(valueS, startPow);
62
63      std::cout << "Again?\n";
64      std::cin >> response;
65
66      if (response == "yes") {
67          main();
68      }
69      else {
70          return 0;
71      }
72      return 0;
73  }
74  int main()
75  {
76      askQuestions();
77  }
```

nascardriver
July 19, 2019 at 6:01 am · Reply

Please edit your comments instead of deleting and re-posting. Code tags work after refreshing the page.

Using `main` causes undefined behavior (Line 21, 59). You can't call `main`. You could instead call `askQuestions`. That isn't good either, but at least it's not UB. You'll learn about loops later.

You fixed some things and added a couple of new issues.

- `main`: Missing return-statement.
- Line 55: Use ++prefix instead of postfix++ unless you need postfix++.
- Line 20: You can't have 2 characters in 1 character.
- `askQuestions`'s return value is unused. Declare it `void`.
- Don't mix `std::endl` and '\n' unless you have a reason to.
- Line 47: Your program only supports numbers up to 128.

Judging by all the whitespace that went missing, I guess you auto-formatted your code. See if you can configure your editor to keep empty lines, because as it is, your code isn't easy to read.

Try applying my suggestions to make sure you understand how to do it properly. Feel free to post your revised version again.

Alexander S.
July 19, 2019 at 6:10 am · Reply

Sorry for the deleting, thought they didn't refresh.

line 20 was a mistake when I tried reediting it so I didn't have to repaste everything.

Then the whitespace was because I repasted it after copying it from here and I noticed it after I posted and was too lazy to redo it again.

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

int main();

int powerCalc(int power) {
    int times{ 0 };
    while (power >= 1) {
        power /= 2;
        ++times;
    }

    return times;
}

void negativeBinary(int value) {

}

void decimalBinary(int value, int startPower) {
    int currentPower{ startPower };
    int currentValue{ value };
    int loopTimes{ powerCalc(startPower) };
    std::string binary{};

    if (value == 0) {
        std::cout << "0 Error \n";

    } else if (value < 0) {
        negativeBinary(value);

    } else {
        for (int i{ 0 }; i < loopTimes; ++i) {
            if (i % 4 == 0 && i != 0) {
                binary += ' ';

            }

            if (currentValue >= currentPower) {
                binary += '1';
                currentValue -= currentPower;

            } else {
                binary += '0';

            }

            currentPower /= 2;

        }
    }
    std::cout << binary << '\n';

}

void askQuestions() {
    int valueS{};
```

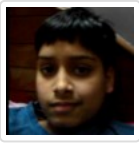
```
58      int startPow{};
59      std::string response{};
60      std::cout << "Please enter an 8-bit integer to convert to decimal\n";
61      std::cin >> valueS;
62      if (valueS > 128) {
63          std::cout << "Too Large of a Number";
64          return;
65      }
66      std::cout << "Please enter a power of 2 (Type '0' for list)\n";
67      std::cin >> startPow;
68      if (startPow == 0) {
69          std::cout << "Powers of 2 \n----------\n";
70          for (int i{ 0 }; i < 15; ++i) {
71              std::cout << std::pow(2, i) << std::endl;
72          }
73          std::cout << "----------\n";
74      }
75      decimalBinary(valueS, startPow);
76
77      std::cout << "Again?\n";
78      std::cin >> response;
79
80  }
81
82
83  int main()
84  {
85      askQuestions();
86      return 0;
87  }
```

nascardriver
July 19, 2019 at 6:15 am · Reply

Well, removing the problematic code is one way of getting rid of the problems :D
Line 71 is still using `std::endl`. Rest looks good now!
Line 62: *up to 127, sorry.

Alexander S.
July 19, 2019 at 6:19 am · Reply

Idk what to do instead of recalling the functions if it causes problems, lol. Easier fix I guess.
I suck at remembering max values lol



kanishkaditya shukla
July 6, 2019 at 8:35 am · Reply

hey nascardriver,

A problem arose when I was trying to implement your suggestion of moving 'int y' into main function(in my previous code given below).

What happened:-
if I move 'int y' into main function and send it to 'binconverter()' as argument then I can't change the

value of 'y' using the statement in 'binconverter()' function [y=y-pow(2,(power-1));],
it simply won't work as 'y' is sent as an argument and assigned to a perimeter in
'binconverter()' the perimeter runs out of scope for main function as the function ends.
and the changed value of 'y' is discarded as the command return to main function.hence,for the next statement
in main function(which needed the changed value), 'y' uses the same value as perimeter again and again as perimeter
and hence, results in incorrect answer.

please let me know if I am wrong(and whats wrong), also if I am right then please suggest an alternative
method of fixing this.
sorry for annoying you.

nascardriver
July 6, 2019 at 8:53 am · Reply

You're right. Print in `binconverter` and `return y`, have a look at the solution if you don't know how to do it. You'll later learn about references, they'll allow functions to make persistent changes to arguments.

kanishkaditya shukla
July 6, 2019 at 7:12 pm · Reply

Thanks got it now.
Tried it and it worked perfectly.
Thanks

kanishkaditya shukla
July 5, 2019 at 9:31 am · Reply

hey alex and nascardriver,I hope you guys are having a good day.
NOTE:-i have used "using namespace std" in the program irrespective of the best practice (sorry for that) in order to keep the project short(however it didn't seem to work to be honest :-)).

HEADER.H

```
#ifndef HEADER_H_INCLUDED
#define HEADER_H_INCLUDED

int input(int power);
char binconverter(int power);

#endif // HEADER_H_INCLUDED
```

INPUT.CPP

```
#include <iostream>
# include "header.h"
# include <cmath>
using namespace std;

int input(int power)
{int x;
   cout << "please enter the no. to be converted into binary\n";
   cin>> x;
```

```
  if(x>=0)
   {return x;}
   else{return ((pow(2,power))+x);};
}
```

BIN CONVERTER.CPP

```
#include <iostream>
# include <cmath>
# include "header.h"
using namespace std;

int y{input(8)};

char binconverter(int power)
{

   if(y<=pow(2,power)&&y>=pow(2,(power-1)))
   {
      y=y-pow(2,(power-1));
      return '1';
   }
   else {
      return '0';
   };
}
```

MAIN.CPP

```
#include <iostream>
# include "header.h"

using namespace std;

int main()
{
 int startpower{8};
  /*cout <<"please enter how many bits do you want\n";
 cin>> startpower;*/
      char a{ binconverter(startpower)};
    --startpower;
      char b{ binconverter(startpower)};
     --startpower;
      char c{ binconverter(startpower)};
     --startpower;
      char d{ binconverter(startpower)};
     --startpower;
      char e{ binconverter(startpower)};
     --startpower;
      char f{binconverter(startpower)};
     --startpower;
      char g{ binconverter(startpower)};
     --startpower;
      char h{ binconverter(startpower)};
      cout<<a<<b<<c<<d<<" "<<e<<f<<g<<h;
   return 0;
}
```

hey nascardriver can you help me to figure out how can i take the user input for the number of bit and create as many as variables needed in order to assign them the value and print it.

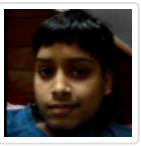
FOR EXAMPLE:- if I want to convert a decimal number into a 16 bit binary number then in order to do this (in my program) I will have to create 16 variables(a,b,c....)and the program will become long,is there any shorter way of doing this so that the program automatically creates variables when needed.

nascardriver
July 5, 2019 at 9:37 am · Reply

Please use code tags.

- Inconsistent formatting. Use your editor's auto-formatting feature.
- Use `std::pow` instead of `pow`.
- Don't call functions from outside of functions. (`input`).
- The last character your program prints should be a line feed to prevent mushed lines.

You're looking for dynamically allocated arrays. They're covered later.



kanishkaditya shukla
July 5, 2019 at 9:59 pm · Reply

Hey nascardriver ,here are some question related to your above reply:-

1. what do you mean by "Don't call functions from outside of functions. (`input`)"?
   whats the problem in doing so?,
   what should i do to remove this error from above code? please help.

2. I didn't understand your fourth suggestion,please help me with that .

Thanks for your reply .
sorry, if I am annoying you, but please help me understand these topics .
(As I am working quite hard to become perfect in these topics.)

It would be very helpful if you explain these question using example (where required) instead of using some scientific word(ex:-mushed lines) as I don't know most of those yet.

nascardriver
July 6, 2019 at 2:46 am · Reply

"mushed" is informal, it's when you take something and make it small, eg. mashed potatoes. The next line printed in the console will be appended to the last line you printed.

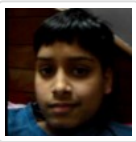
> whats the problem in doing so?
You don't know the order global variables get initialized in. It could happen that `input` tries to access something that doesn't exist yet.

> what should i do to remove this error from above code?
Move `y` into `main` and pass it to `binconverter`.

> The last character your program prints should be a line feed

```
1  int main(void)
2  {
3    std::cout << "Hello"; // bad
4    return 0;
5  }
6
7  int main(void)
8  {
```

```
  std::cout << "Hello\n"; // good
  return 0;
}

int main(void)
{
  int a{ 123 };

  std::cout << a; // bad

  return 0;
}

int main(void)
{
  int a{ 123 };

  std::cout << a << '\n'; // good

  return 0;
}
```



kanishkaditya shukla
July 6, 2019 at 2:54 am · Reply

Thanks for the reply.
I understood every solution now .

timbo
July 3, 2019 at 1:48 pm · Reply

i made a program, converts inetegers to 8 bit binary, twos complement if negative

```
char intToChar(int n)
{
    //must be an integer
    char c = n + 48;
    return c;
}

void intToBinary(char s[], int a)
{
    int tempA = a;
    int mod = 0;
    //set the copied inputted number to the opposite if negative
    if (a < 0)
    {
        tempA = -a;
    }
    //set to 0 if inputted number is 0
    else if (a == 0)
    {
        for (int i = 0; i < 8; i++)
        {
            s[i] = '0';
        }

        s[8] = '\0';
    }
```

```
    //if input is 0 dont do anything
    if (a == 0)
        int heehee = 0;
    else
    {

        int power2 = 1;
        /*
        int b = 8;
        //find out where to start putting digits
        while (tempA >= power2 || b == 0)
        {
            power2 *= 2;
            b--;
        }
        */
        int i2 = 7;
        //convert to unsigned binary
        while (tempA != 0)
        {
            mod = tempA % 2;
            tempA = tempA / 2;
            s[i2] = intToChar(mod);
            i2--;
        }
        s[8] = '\0';
        //set remaining digits to 0
        for (; i2 >= 0; i2--)
            s[i2] = '0';
        //do steps only if original number was negative
        if (a < 0)
        {
            //invert digits
            for (int i4 = 0; i4 < 8; i4++)
            {
                s[i4] = (s[i4] == '1') ? '0' : '1';
            }
            //add one
            bool isCarryover = false;
            int i5 = 7;
            do
            {
                isCarryover = false;
                s[i5] += 1;

                if (s[i5] > 49)
                {
                    s[i5] = '0';
                    isCarryover = true;
                }
                i5--;
            } while (isCarryover);

        }

    }

}

int main()
{
    int in;
```

```
    char str[81];

    std::cout << "integer to 8-bit binary (twos complement representation)\n";
    std::cout << "============================================================\n\n";
    std::cout << "enter an integer from -128 to 127:\n\t";

    std::cin >> in;
    intToBinary(str, in);
    std::cout << "\n\nbinary of " << in << " is " << str                    ;
    if (in < 0)
        std::cout << " (two's complement)";
    std::cout << '\n';

    return 0;
}
```

Matthew E
June 19, 2019 at 6:54 pm · Reply

I have created a very fancy solution :)
funcs.h is just my getInteger() function.

```
#include <iostream>
#include "funcs.h"

int main() {
    int x{ getInteger() };

    std::cout << (x >= 128);
    x = ((x >= 128) ? x - 128 : x);

    std::cout << (x >= 64);
    x = ((x >= 64) ? x - 64 : x);

    std::cout << (x >= 32);
    x = ((x >= 32) ? x - 32 : x);

    std::cout << (x >= 16);
    x = ((x >= 16) ? x - 16 : x);

    std::cout << ' ';

    std::cout << (x >= 8);
    x = ((x >= 8) ? x - 8 : x);

    std::cout << (x >= 4);
    x = ((x >= 4) ? x - 4 : x);

    std::cout << (x >= 2);
    x = ((x >= 2) ? x - 2 : x);

    std::cout << (x >= 1);

    return 0;
}
```

« Older Comments 1 ... 5 6 7