Binary & Hex
+ Bitwise, Logical & Shift Operators in C
SOLUTIONS

1. 110101_2 => $\mathbf{53_{10}}$

   10110111_2 => $\mathbf{183_{10}}$

2. 41_{10} => $\mathbf{101001_2}$ *# Can also write 0b101001*

   123_{10} => $\mathbf{1111011_2}$ *# Can also write 0b1111011*

3. Suppose x = 0xba and y = 0x2d. Do the following calculations by converting these hex numbers to bits and then expressing the resulting bits in hex.

   Note: x = 0xba = **0b10111010** (= 186_{10}) and y = 0x2d = **0b00101101** (= 45_{10}). (Decimal equivalents are **not** necessary, but show that addition/subtraction below work correctly.)

   ```
   x + y = 0b10111010 + 0b00101101 = 0b11100111 = 0xe7 (= 231₁₀)
   x - y = 0b10111010 - 0b00101101 = 0b10001101 = 0x8d (= 141₁₀)
   x & y = 0b10111010 & 0b00101101 = 0b00101000 = 0x28
   x | y = 0b10111010 | 0b00101101 = 0b10111111 = 0xbf
   x ^ y = 0b10111010 | 0b00101101 = 0b10010111 = 0x97
   (~x) ^ y = 0b01000101 | 0b00101101 = 0b01101000 = 0x68
   ```

   Note: You can check your answers using
   https://www.programiz.com/c-programming/online-compiler/
   with the following C program (appropriately edited)

   ```
   #include <stdio.h>

   int main() {
         char x = 0xba; // Use char type for 1-byte values
         char y = 0x2d; // Use char type for 1-byte values
         printf("Result is 0x%hhx", x + y);
           // %hhx displays byte values as 2 hex digits,
   }
   ```

4. Recall that a C int is a 4-byte (32 bit) signed integer. Suppose the following ints are defined:

```
int zero = 0x0;
int five = 0x5;
int six = 0x6;
int ten = 0xa;
```

Determine the results of the following in 4-byte hex (no need to show leading zeros). Recall that the logical operators **&&**, **||**, and **!** treat zero as **false**, any non-zero number as **true**, and always return one of **`0x0`** (for **false**) and **`0x1`** (for **true**). As in Problem 3, you can check your answers using the online C compiler (but use `int` rather than `char` for your value types).

Expression	Value	Expression	Value
`five & ten`	**`0x0`**	`five && ten`	**`0x1`**
`six & ten`	**`0x2`**	`six && ten`	**`0x1`**
`six & zero`	**`0x0`**	`six && zero`	**`0x0`**
`five | ten`	**`0xf`**	`five || ten`	**`0x1`**
`six | ten`	**`0xe`**	`six || ten`	**`0x1`**
`zero | zero`	**`0x0`**	`zero || zero`	**`0x0`**
`five ^ ten`	**`0xf`**		
`six ^ ten`	**`0xc`**		
`~zero`	**`0xffffffff`**	`!zero`	**`0x1`**
`~six`	**`0xfffffff9`**	`!six`	**`0x0`**
`six << 1`	**`0xc`**	`(~six) << 1`	**`0xfffffff2`**
`six << 2`	**`0x18`**	`(~six) << 2`	**`0xffffffe4`**
`six >> 1`	**`0x3`**	`(~six) >> 1`	**`0xfffffffc`**
`six >> 2`	**`0x1`**	`(~six) >> 2`	**`0xfffffffe`**