Swap Bits Program Explanation

Key Points:

1. a ^ 0 = a
2. a ^ a = 0
3. a & a = a
4. a & 0 = 0
5. 0 | 0 = 0
6. 0 | 1 = 1

Question:

1. ### Aim:

   A positive number is given as input along with two positions p1, p2 and length n. Your job is to write a program that swaps 'n' bits of the number starting from positions p1, p2. p1, p2 are 0-indexed values that start from least significant bit of the number.

   Use unsigned int data type.

2. ### Explanation:

   Sample 1:

   Number = 47. Its binary representation is 00101111

   p1 = 1 (Start from 2nd bit from right side)

   p2 = 5 (Start from 6th bit from right side)

   n = 3 (No of bits to be swapped)

   The 3 bits starting from p1 are 0010'111'1 and the 3 bits staring from p2 are '001'01111. After swapping these two, the resultant binary string will be 11100011 which is equal to decimal value 227.

   Hence the output must be 227.

   Sample 2:

   Number = 28. Its binary representation is 11100

   p1 = 0 (Start from 1st bit from right side)

   p2 = 3 (Start from 4th bit from right side)

n = 2 (No of bits to be swapped)

The 2 bits starting from p1 are 111'00' and the 2 bits staring from p2 are '11'100. After swapping these two, the resultant binary string will be 00111 which is equal to decimal value 7.

Hence the output must be 7.

Sample 3:

Number=178. Its binary representation is 10110010.

p1 = 0 (Start from 1st bit from right side)

p2 = 30 (Start from 31st bit from right side)

n = 5 (No of bits to be swapped)

In this case, since range exceeds at position2, use 0's in that case. So, the bits starting from p1 are '10010' and the bits staring from p2 are '00000'.

After swapping the two, the resultant binary string will be '10010000000000000000000000010100000'. But considering only 32 bits, discard extra bits on left side and hence resultant string will be '10000000000000000000000010100000' which is equal to decimal value 2147483808

Hence the output must be 2147483808.

3. Constraints:

   i. The positions and the length 'n' must always be positive.

   ii. 0<=p1<p2<=32.

   iii. If the range overflows 32, discard the extra bits. (See Sample 3)

   iv. It is guaranteed that positions and ranges will not overlap.

   v. It is guaranteed that the number will not exceed 32-bit range.

4. *Note:*

   i. If constraints are violated, then return 0.

   ii. Please Use 32-bit data (unsigned int) type to get accurate results.

Program:

```
#include<stdio.h>

int swapBits(unsigned int number, unsigned int p1, unsigned int p2, unsigned int n)
{
   unsigned int a = (number >> p1) & ((1U << n) - 1);

   unsigned int b = (number >> p2) & ((1U << n) - 1);

   unsigned int c = (a ^ b);

   c = (c << p1) | (c << p2);

   unsigned int result = number ^ c;

   return result;
}
```

Steps:

1.Extraction:

In order to extract the bits to be swapped, we shift them to rightmost side. As we know, **a & 1 = a** and **a & 0 = 0**. We now, create a binary number with exactly 'n' 1's and remaining as 0's because when we perform the **AND (&)** operation, we only get the bits to be swapped.

From the above example, number = 28, p1 = 0, p2 = 3 and n = 2.

number >> p1 → 11000 >> 0 → 11000

number >> p2 → 11000 >> 3 → 00011

1U << n → 100 [Left-Shift Operation] //U stands for unsigned

1U << n – 1 → 100 - 1 (in binary) → 011 [1's complement]

unsigned int a = number >> p1 & ((1U << n) – 1);

```
          11100
AND (&)   00011
          --------
           00000      ['00' is extracted]
          --------
```

```
unsigned int a = number >> p1 & ((1U << n) – 1);
```

```
        00011
AND (&) 00011
        --------
        00011       ['11' is extracted]
        --------
```

2. Calculating Xor of ‘a’ and ‘b’:

We calculate the **XOR (^)** of 'a' & 'b' and store it in another unsigned int variable c. The reason for this step will be explained later.

```
unsigned int c = (a ^ b);
```

```
        00000
XOR (^) 00011
        --------
        00011
        --------
```

3. Substitution:

We try to substitute the **XOR (^)** values of 'a' and 'b' in the place of bits that are to be swapped by shifting the bits of c by p1 and p2 simultaneously. We then, perform **OR (|)** operation in order to combine them as a single value.

c << p1 → 00011 << 0 → 00011

c << p2 → 00011 << 3 → 11000

```
c = c << p1 | c << p2;
```

```
        00011
OR (|)  11000
        --------
        11011       [Value of c is substituted]
        --------
```

4.Calculating result:

As we know that **a ^ b ^ a = b** and **a ^ b ^ b = a**, to get the result we perform **XOR (^)** operation for the given number and the value of 'c'. Since, 'c' contains the result of **a ^ b**, when we perform **XOR (^)** operation, the values of 'a' and 'b' will get swapped.

unsigned int result = number ^ c;

```
    11100
XOR (^) 11011
   --------
    00111
   ---------
```

Return the result.

[The End]