Bit Manipulations

(Part -II)

1) Bit Masking

int masked_value = x & mask;

2) Setting Multiple Bits:

#define MASK 0x0F
x |= MASK; // Set lower 4 bits

3) Clearing Multiple Bits:

#define MASK 0xF0
x &= ~MASK; // Clear upper 4 bits

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, mask;

    // 1) Bit Masking
    printf("Enter an integer (x): ");
    scanf("%d", &x);
    printf("Enter a mask: ");
    scanf("%d", &mask);

    int masked_value = x & mask;
    printf("\nBit Masking: %d (%s) AND %d (%s) = %d (%s)\n", x, decimalToBinaryWithSpace(x), mask, decimalToBinaryWithSpace(mask), masked_value, decimalToBinaryWithSpace(masked_value));

    // 2) Setting Multiple Bits
    #define MASK_SET 0x0F
    x |= MASK_SET;
    printf("\nSetting Multiple Bits: %d (%s)\n", x, decimalToBinaryWithSpace(x));

    // 3) Clearing Multiple Bits
    #define MASK_CLEAR 0xF0
    x &= ~MASK_CLEAR;
    printf("Clearing Multiple Bits: %d (%s)\n", x, decimalToBinaryWithSpace(x));

    return 0;
}
```

Output:

```
~ ./masking_set_multiple
Enter an integer (x): 337
Enter a mask: 3

Bit Masking: 337 ( 0000 0000 0000 0000 0001 0101 0001) AND 3 ( 0000 0000 0000 0000 0000 0000 0011) = 1 ( 0000 0000 0000 0000 0000 0000 0001)

Setting Multiple Bits: 351 ( 0000 0000 0000 0000 0001 0101 1111)
Clearing Multiple Bits: 271 ( 0000 0000 0000 0000 0001 0000 1111)
```

4) Selective Bit Inversion:

Invert specific bits (useful in scenarios like manipulating control registers):
x ^= MASK;

5) Compute the Sign of an Integer:
int sign = (x >> 31) | !!x;

6) Conditional Negation:

int r = (x >> 31 & (y ^ z)) ^ z;

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y, z, MASK;

    // 4) Selective Bit Inversion
    printf("Enter an integer (x): ");
    scanf("%d", &x);
    printf("Enter a mask for inversion: ");
    scanf("%d", &MASK);

    x ^= MASK;
    printf("\nSelective Bit Inversion: %d (%s)\n", x, decimalToBinaryWithSpace(x));

    // 5) Compute the Sign of an Integer
    int sign = (x >> 31) | !!x; // !!x converts x to 0 (for x=0) or 1 (for x!=0)
    printf("\nSign of %d: %d\n", x, sign);

    // 6) Conditional Negation
    printf("Enter two more integers (y and z): ");
    scanf("%d %d", &y, &z);

    int r = (x >> 31 & (y ^ z)) ^ z;
    printf("\nConditional Negation Result: %d (%s)\n", r, decimalToBinaryWithSpace(r));

    return 0;
}
```

Output:

```
~ ./inversion_computeSign_negation
Enter an integer (x): 552
Enter a mask for inversion: 12

Selective Bit Inversion: 548 ( 0000 0000 0000 0000 0010 0010 0100)

Sign of 548: 1
Enter two more integers (y and z): 2336 2251

Conditional Negation Result: 2251 ( 0000 0000 0000 0000 1000 1100 1011)
```

7) Determine if two integers have the same sign:

bool sameSign = (x ^ y) >= 0;

8) Isolate the rightmost bit that is set to 1:

int rightmostBit = x & (-x);

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> // For bool data type

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y;

    // 7) Determine if two integers have the same sign
    printf("Enter two integers (x and y): ");
    scanf("%d %d", &x, &y);

    bool sameSign = (x ^ y) >= 0;
    printf("\nDo %d and %d have the same sign? %s\n", x, y, sameSign ? "Yes" : "No");

    // 8) Isolate the rightmost bit that is set to 1
    printf("\nEnter an integer (x) to isolate its rightmost set bit: ");
    scanf("%d", &x);

    int rightmostBit = x & (-x);
    printf("\nRightmost set bit of %d: %d (%s)\n", x, rightmostBit, decimalToBinaryWithSpace(rightmostBit));

    return 0;
}
```

Output:

```
~ ./sameSign_IsolateRightMostSeto1
Enter two integers (x and y): 25 23

Do 25 and 23 have the same sign? Yes

Enter an integer (x) to isolate its rightmost set bit: 2263

Rightmost set bit of 2263: 1 ( 0000 0000 0000 0000 0000 0000 0001)
```

9) Merge bits from two values according to a mask:

int result = (x & mask) | (y & ~mask);

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y, mask;

    // Input the values
    printf("Enter the first integer (x): ");
    scanf("%d", &x);
    printf("Enter the second integer (y): ");
    scanf("%d", &y);
    printf("Enter a mask: ");
    scanf("%d", &mask);

    // 9) Merge bits from two values according to a mask
    // The operation (x & mask) retains the bits in x where the mask has 1s.
    // The operation (y & ~mask) retains the bits in y where the mask has 0s.
    // The final OR operation combines the retained bits from x and y to produce the result.
    int result = (x & mask) | (y & ~mask);
    printf("\nMerged result of %d (%s) and %d (%s) with mask %d (%s) is: %d (%s)\n", x, decimalToBinaryWithSpace(x), y, decimalToBinaryWithSpace(y), mask, decimalToBinaryWithSpace(mask), result, decimalToBinaryWithSpace(result));

    return 0;
}
```

Output:

```
~ ./merge_bits_2values
Enter the first integer (x): 6691
Enter the second integer (y): 3302
Enter a mask: 2

Merged result of 6691 ( 0000 0000 0000 0001 1010 0010 0011) and 3302 ( 0000 0000 0000 0000 1100 1110 0110) with mask 2 ( 0000 0000 0000 0000 0000 0000 0010) is: 3302 ( 0000 0000 0000 0000 1100 1110 0110)
```

10) Detect if a number has consecutive bits set:

bool hasConsecutiveOnes = (x & (x << 1)) != 0;

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> // For bool data type

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x;

    // Input the value
    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // By shifting x to the left by 1, we move all bits one position to the left.
    // If x has two consecutive bits set, then x & (x << 1) will have at least one bit set.
    // For example, consider x = ...011... (with 2 consecutive set bits). After left shifting, it becomes ...110...
    // When we perform bitwise AND between x and (x << 1), it results in ...010... which is not zero.
    bool hasConsecutiveOnes = (x & (x << 1)) != 0;
    printf("\nDoes %d (%s) have consecutive set bits? %s\n", x, decimalToBinaryWithSpace(x), hasConsecutiveOnes ? "Yes" : "No");

    return 0;
}
```

Output:

```
~ ./detect_consecutive_bits
Enter an integer (x): 7841

Does 7841 ( 0000 0000 0000 0001 1110 1010 0001) have consecutive set bits? Yes
```

11) Count Set Bits

```
count = 0;
while(x) {
   count += x & 1;
   x >>= 1;
}
```

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x;

    // Input the value
    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // This algorithm works by repeatedly checking the least significant bit of x and then right shifting x.
    // The operation (x & 1) checks if the least significant bit of x is set.
    // If it's set, it adds 1 to the count.
    // The operation (x >>= 1) right shifts x, effectively dropping its least significant bit.
    // This process continues until x becomes zero.
    int count = 0;
    int originalX = x;  // Store the original value of x for later use
    while(x) {
        count += x & 1;
        x >>= 1;
    }
    printf("\nNumber of set bits in %d (%s) is: %d\n", originalX, decimalToBinaryWithSpace(originalX), count);

    return 0;
}
```

Output:

```
~ ./count_set_bits
Enter an integer (x): 295841

Number of set bits in 295841 ( 0000 0000 0100 1000 0011 1010 0001) is: 7
```

12) Swap Bits

```
if (((x >> p) & 1) != ((x >> q) & 1)) {
    x ^= (1 << p) | (1 << q);
}
```

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, p, q;

    // Input the values
    printf("Enter an integer (x): ");
    scanf("%d", &x);
    printf("Enter position of first bit to swap (p): ");
    scanf("%d", &p);
    printf("Enter position of second bit to swap (q): ");
    scanf("%d", &q);

    // First, we check if the bits at position p and q are different.
    // If they are different, then we toggle both bits.
    // To check if they are different, we shift x right by p and q positions and check the least significant bit.
    // To toggle the bits, we use XOR (^) with (1 << p) and (1 << q).
    if (((x >> p) & 1) != ((x >> q) & 1)) {
        x ^= (1 << p) | (1 << q);
    }

    printf("\nValue of %d (%s) after swapping bits at positions %d and %d is: %d (%s)\n", x, decimalToBinaryWithSpace(x), p, q, x, decimalToBinaryWithSpace(x));

    return 0;
}

```

Output:

```
~ ./swap_bits
Enter an integer (x): 268541
Enter position of first bit to swap (p): 9
Enter position of second bit to swap (q): 15

Value of 268541 ( 0000 0000 0100 0001 1000 1111 1101) after swapping bits at positions 9 and 15 is: 268541 ( 0000 0000 0100 0001 1000 1111 1101)
```

13) Detect if Two Integers have Opposite Signs

bool opposite = (x ^ y) < 0;

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> // For bool data type

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y;

    // Input the values
    printf("Enter the first integer (x): ");
    scanf("%d", &x);
    printf("Enter the second integer (y): ");
    scanf("%d", &y);

    // We use the XOR (^) operation on x and y.
    // If the result is negative, it implies that x and y have opposite signs.
    // This is because the sign bit (the leftmost bit) of the two numbers is different.
    bool opposite = (x ^ y) < 0;
    printf("\nDo %d (%s) and %d (%s) have opposite signs? %s\n", x, decimalToBinaryWithSpace(x), y, decimalToBinaryWithSpace(y), opposite ? "Yes" : "No");

    return 0;
}
```

Output:

```
~ ./detect_opp_sign
Enter the first integer (x): 891
Enter the second integer (y): -692

Do 891 ( 0000 0000 0000 0000 0011 0111 1011) and -692 ( 1111 1111 1111 1111 1101 0100 1100) have opposite signs? Yes
```

14) Get Absolute Value without Branching

int mask = x >> (sizeof(int) * 8 - 1);
int abs_val = (x + mask) ^ mask;

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x;

    // Input the value
    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // The operation (x >> (sizeof(int) * 8 - 1)) will produce -1 if x is negative and 0 if x is positive.
    // If x is negative, adding -1 will decrement x, and XOR operation with -1 will flip all bits (equivalent to two's complement).
    // If x is positive, adding 0 won't change x, and XOR operation with 0 won't change x either.
    int mask = x >> (sizeof(int) * 8 - 1);
    int abs_val = (x + mask) ^ mask;
    printf("\nAbsolute value of %d (%s) is: %d (%s)\n", x, decimalToBinaryWithSpace(x), abs_val, decimalToBinaryWithSpace(abs_val));

    return 0;
}

```

Output:

```
~ ./absolute_value_without_branching
Enter an integer (x): -9854

Absolute value of -9854 ( 1111 1111 1111 1101 1001 1000 0010) is: 9854 ( 0000 0000 0000 0010 0110 0111 1110)
```

15) Check if a number is even or odd:

```
if (x & 1) {
   // odd
} else {
   // even
}
```

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x;

    // Input the value
    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // The operation (x & 1) checks the least significant bit of x.
    // If the least significant bit is set (1), then the number is odd.
    // If the least significant bit is not set (0), then the number is even.
    if (x & 1) {
        printf("\n%d (%s) is odd.\n", x, decimalToBinaryWithSpace(x));
    } else {
        printf("\n%d (%s) is even.\n", x, decimalToBinaryWithSpace(x));
    }

    return 0;
}
```

Output:

```
~ ./even_or_odd
Enter an integer (x): 2513

2513 ( 0000 0000 0000 0000 1001 1101 0001) is odd.
~ ./even_or_odd
Enter an integer (x): 2514

2514 ( 0000 0000 0000 0000 1001 1101 0010) is even.
```

16) Find Minimum or Maximum without branching:

int y;
int minimum = y ^ ((x ^ y) & -(x < y));
int maximum = x ^ ((x ^ y) & -(x < y));

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y;

    // Input the values
    printf("Enter the first integer (x): ");
    scanf("%d", &x);
    printf("Enter the second integer (y): ");
    scanf("%d", &y);

    // The expression (x < y) returns 1 if true, 0 if false.
    // Taking the negative of this result gives -1 if x < y, 0 otherwise.
    // The operation (x ^ y) gives a value that has 1s in the positions where x and y are different.
    // ANDing this with the negative of the comparison result effectively chooses y if x < y, x otherwise.
    // Finally, XORing with y or x gives the minimum or maximum value respectively.
    int minimum = y ^ ((x ^ y) & -(x < y));
    int maximum = x ^ ((x ^ y) & -(x < y));

    printf("\nBetween %d (%s) and %d (%s):\n", x, decimalToBinaryWithSpace(x), y, decimalToBinaryWithSpace(y));
    printf("Minimum is: %d (%s)\n", minimum, decimalToBinaryWithSpace(minimum));
    printf("Maximum is: %d (%s)\n", maximum, decimalToBinaryWithSpace(maximum));

    return 0;
}

```

Output:

```
~ ./min_max_no_branch
Enter the first integer (x): 4458
Enter the second integer (y): 2360

Between 4458 ( 0000 0000 0000 0001 0001 0110 1010) and 2360 ( 0000 0000 0000 0000 1001 0011 1000):
Minimum is: 2360 ( 0000 0000 0000 0000 1001 0011 1000)
Maximum is: 4458 ( 0000 0000 0000 0001 0001 0110 1010)
```

17) Swap two numbers without a temporary variable:

x = x ^ y;
y = x ^ y;
x = x ^ y;

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y;

    // Input the values
    printf("Enter the first integer (x): ");
    scanf("%d", &x);
    printf("Enter the second integer (y): ");
    scanf("%d", &y);

    printf("\nBefore swapping: x = %d (%s), y = %d (%s)\n", x, decimalToBinaryWithSpace(x), y, decimalToBinaryWithSpace(y));

    // The XOR operation can be used to swap two numbers without a temporary variable.
    // The idea is that XOR of a number with itself is 0, and XOR of a number with 0 is the number itself.
    // So, by performing XOR operations sequentially, we can effectively swap the values of x and y.
    x = x ^ y;
    y = x ^ y;  // At this point, y has the original value of x
    x = x ^ y;  // At this point, x has the original value of y

    printf("\nAfter swapping: x = %d (%s), y = %d (%s)\n", x, decimalToBinaryWithSpace(x), y, decimalToBinaryWithSpace(y));

    return 0;
}
```

Output:

```
~ ./swap_no_remp_var
Enter the first integer (x): 4581
Enter the second integer (y):
-895

Before swapping: x = 4581 ( 0000 0000 0000 0001 0001 1110 0101), y = -895 ( 1111 1111 1111 1111 1100 1000 0001)

After swapping: x = -895 ( 1111 1111 1111 1111 1100 1000 0001), y = 4581 ( 0000 0000 0000 0001 0001 1110 0101)
```

18) Compute modulus division by a power of 2:

int divisor = 4; // (which is 2^2)
int result = x & (divisor - 1);

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x;

    // Input the value of x
    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // When we want to compute the modulus of a number by a power of 2 (like 2, 4, 8, 16, ...),
    // we can use a quick trick with bitwise operations.
    // This method involves ANDing the number with (power of 2 - 1).
    // The result of this operation gives the remainder when the number is divided by the power of 2.
    int divisor = 4;  // (which is 2^2)
    int result = x & (divisor - 1);

    printf("\n%d (%s) modulus %d is: %d (%s)\n", x, decimalToBinaryWithSpace(x), divisor, result, decimalToBinaryWithSpace(result));

    return 0;
}
```

Output:

```
~ ./modulus_power_2
Enter an integer (x): -9568

-9568 ( 1111 1111 1111 1101 1010 1010 0000) modulus 4 is: 0 ( 0000 0000 0000 0000 0000 0000 0000)
~ ./modulus_power_2
Enter an integer (x): 2514

2514 ( 0000 0000 0000 0000 1001 1101 0010) modulus 4 is: 2 ( 0000 0000 0000 0000 0000 0000 0010)
~ ./modulus_power_2
Enter an integer (x): 36923

36923 ( 0000 0000 0000 1001 0000 0011 1011) modulus 4 is: 3 ( 0000 0000 0000 0000 0000 0000 0011)
```

19) Find average of two numbers without overflow:

int average = (x & y) + ((x ^ y) >> 1);

Program for the above:

```c
#include <stdio.h>
#include <stdlib.h>

// Function to convert decimal to binary with spaces for every 4 bits
char* decimalToBinaryWithSpace(int num) {
    char* binary = (char*)malloc(36);  // 32 bits + 3 spaces + null terminator for an int
    binary[35] = '\0';
    int index = 34;

    for(int i = 0; i < 32; i++) {
        if (i > 0 && i % 4 == 0) {
            binary[index--] = ' '; // Add space every 4 bits
        }
        if((num & 1) == 1) {
            binary[index] = '1';
        } else {
            binary[index] = '0';
        }
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, y;

    // Input the values of x and y
    printf("Enter the first integer (x): ");
    scanf("%d", &x);
    printf("Enter the second integer (y): ");
    scanf("%d", &y);

    // Calculating the average directly as (x+y)/2 can cause overflow if x and y are both large.
    // This method avoids overflow by focusing on common bits (x & y) and bits where they differ ((x ^ y) >> 1).
    // The result of these operations is then added to get the average.
    int average = (x & y) + ((x ^ y) >> 1);

    printf("\nThe average of %d (%s) and %d (%s) is: %d (%s)\n", x, decimalToBinaryWithSpace(x), y, decimalToBinaryWithSpace(y), average, decimalToBinaryWithSpace(average));

    return 0;
}
```

Output:

```
~ ./averge_without_overflow
Enter the first integer (x): 45812
Enter the second integer (y): 33692

The average of 45812 ( 0000 0000 0000 1011 0010 1111 0100) and 33692 ( 0000 0000 0000 1000 0011 1001 1100) is: 39752 ( 0000 0000 0000 1001 1011 0100 1000)
```

20) Check if all bits in a number are 0 (number is zero):

bool isZero = !x;

21) Check if only a single bit is set in a number:

bool isSingleBitSet = x && !(x & (x-1));

Program for the above:

```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    int x;

    // Input the value of x
    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // If x is 0, then !x will return 1 (true), otherwise it will return 0 (false).
    bool isZero = !x;
    printf("\nIs %d zero? %s\n", x, isZero ? "Yes" : "No");

    // This technique works by checking if x is non-zero (to handle the special case when x = 0)
    // and then determining if x has exactly one bit set.
    // The expression (x & (x-1)) will unset the rightmost set bit in x.
    // If the result is 0, then x had only one bit set.
    bool isSingleBitSet = x && !(x & (x-1));
    printf("Does %d have only a single bit set? %s\n", x, isSingleBitSet ? "Yes" : "No");

    return 0;
}
```

Output:

```
~ ./numberisZero_check_single_bit_set
Enter an integer (x): 0

Is 0 zero? Yes
Does 0 have only a single bit set? No
~ ./numberisZero_check_single_bit_set
Enter an integer (x): 5890

Is 5890 zero? No
Does 5890 have only a single bit set? No
~ ./numberisZero_check_single_bit_set
Enter an integer (x): 1

Is 1 zero? No
Does 1 have only a single bit set? Yes
```

Written By: **Yashwanth Naidu Tikkisetty**

Happy learning.

Learn together, Grow together.

Follow me to receive updates regarding Embedded Systems.

Connect with me on LinkedIn: https://www.linkedin.com/in/t-yashwanth-naidu/

 T Yashwanth Naidu

 https://github.com/T-Yashwanth-Naidu