Bit Manipulations

(Part -I)

1) Bitwise OR
result = x | y;

2) Bitwise NOT
x = ~x;

3) Bitwise AND
result = x & y;

4) Bitwise XOR
result = x ^ y;

Program for the above:

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

char* decimalToBinary(int num) {
    char* binary = (char*)malloc(33);  // 32 bits + null terminator for an int
    binary[32] = '\0';
    int index = 31;

    for(int i = 0; i < 32; i++) {
        if((num & 1) == 1)
            binary[index] = '1';
        else
            binary[index] = '0';
        num >>= 1;
        index--;
    }
    return binary;
}

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

    printf("Enter two integers (x y): ");
    scanf("%d %d", &x, &y);

    // Bitwise OR
    result = x | y;
    printf("Bitwise OR of %d (%s) and %d (%s): %d (%s)\n", x, decimalToBinary(x), y, decimalToBinary(y), result, decimalToBinary(result));

    // Bitwise NOT
    x = ~x;
    printf("Bitwise NOT of x: %d (%s)\n", x, decimalToBinary(x));
    x= ~x;

    // Bitwise AND
    result = x & y;
    printf("Bitwise AND of %d (%s) and %d (%s): %d (%s)\n", x, decimalToBinary(x), y, decimalToBinary(y), result, decimalToBinary(result));

    // Bitwise XOR
    result = x ^ y;
    printf("Bitwise XOR of %d (%s) and %d (%s): %d (%s)\n", x, decimalToBinary(x), y, decimalToBinary(y), result, decimalToBinary(result));

    return 0;
}
```

Output:

```
~ ./basic_bitwise
Enter two integers (x y): 5 6
Bitwise OR of 5 (00000000000000000000000000000101) and 6 (00000000000000000000000000000110): 7 (00000000000000000000000000000111)
Bitwise NOT of x: -6 (11111111111111111111111111111010)
Bitwise AND of 5 (00000000000000000000000000000101) and 6 (00000000000000000000000000000110): 4 (00000000000000000000000000000100)
Bitwise XOR of 5 (00000000000000000000000000000101) and 6 (00000000000000000000000000000110): 3 (00000000000000000000000000000011)
```

5) Left Shift
x = x << n;

6) Right Shift
x = x >> n;

Program for the above:

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

char* decimalToBinary(int num) {
    char* binary = (char*)malloc(33);  // 32 bits + null terminator for an int
    binary[32] = '\0';
    int index = 31;

    for(int i = 0; i < 32; i++) {
        if((num & 1) == 1)
            binary[index] = '1';
        else
            binary[index] = '0';
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, n;

    printf("Enter an integer (x): ");
    scanf("%d", &x);
    printf("Enter number of positions for shifting (n): ");
    scanf("%d", &n);

    // Left Shift
    int leftShiftResult = x << n;
    printf("Left Shift of %d (%s) by %d positions: %d (%s)\n", x, decimalToBinary(x), n, leftShiftResult, decimalToBinary(leftShiftResult));

    // Right Shift
    int rightShiftResult = x >> n;
    printf("Right Shift of %d (%s) by %d positions: %d (%s)\n", x, decimalToBinary(x), n, rightShiftResult, decimalToBinary(rightShiftResult));

    return 0;
}
```

Output:

```
~ ./bit_shifting
Enter an integer (x): 9
Enter number of positions for shifting (n): 2
Left Shift of 9 (00000000000000000000000000001001) by 2 positions: 36 (00000000000000000000000000100100)
Right Shift of 9 (00000000000000000000000000001001) by 2 positions: 2 (00000000000000000000000000000010)
```

7) Setting a Bit (Bitwise OR)
x = x | (1 << n);

8) Clearing a Bit (Bitwise AND)
x = x & ~(1 << n);

9) Toggling a Bit (Bitwise XOR)
x = x ^ (1 << n);

10) Checking a Bit
bit = (x & (1 << n)) != 0;

Program for the above:

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

char* decimalToBinary(int num) {
    char* binary = (char*)malloc(33);  // 32 bits + null terminator for an int
    binary[32] = '\0';
    int index = 31;

    for(int i = 0; i < 32; i++) {
        if((num & 1) == 1)
            binary[index] = '1';
        else
            binary[index] = '0';
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, n;
    int bit;

    printf("Enter an integer (x): ");
    scanf("%d", &x);
    printf("Enter the bit position (0-based index) to manipulate (n): ");
    scanf("%d", &n);

    printf("\nValue of x in binary:%s\n",decimalToBinary(x));

    // Setting a Bit (Bitwise OR)
    int setBitResult = x | (1 << n);
    printf("\nSetting bit %d of %d (%s): %d (%s)\n", n, x, decimalToBinary(x), setBitResult, decimalToBinary(setBitResult));

    // Clearing a Bit (Bitwise AND)
    int clearBitResult = x & ~(1 << n);
    printf("Clearing bit %d of %d (%s): %d (%s)\n", n, x, decimalToBinary(x), clearBitResult, decimalToBinary(clearBitResult));

    // Toggling a Bit (Bitwise XOR)
    int toggleBitResult = x ^ (1 << n);
    printf("Toggling bit %d of %d (%s): %d (%s)\n", n, x, decimalToBinary(x), toggleBitResult, decimalToBinary(toggleBitResult));

    // Checking a Bit
    bit = (x & (1 << n)) != 0;
    printf("Checking bit %d of %d (%s): %s\n", n, x, decimalToBinary(x), bit ? "Set" : "Not Set");

    return 0;
}

```

Output:

```
~ ./set_clear_toggle_check
Enter an integer (x): 23
Enter the bit position (0-based index) to manipulate (n): 3

Value of x in binary:00000000000000000000000000010111

Setting bit 3 of 23 (00000000000000000000000000010111): 31 (00000000000000000000000000011111)
Clearing bit 3 of 23 (00000000000000000000000000010111): 23 (00000000000000000000000000010111)
Toggling bit 3 of 23 (00000000000000000000000000010111): 31 (00000000000000000000000000011111)
Checking bit 3 of 23 (00000000000000000000000000010111): Not Set
~
```

11) Rotate Bits (Left)
int rotate_left = (x << n) | (x >> (sizeof(x)*8 - n));

12) Rotate Bits (Right)
int rotate_right = (x >> n) | (x << (sizeof(x)*8 - n));

Program for above:

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

char* decimalToBinary(int num) {
    char* binary = (char*)malloc(33);  // 32 bits + null terminator for an int
    binary[32] = '\0';
    int index = 31;

    for(int i = 0; i < 32; i++) {
        if((num & 1) == 1)
            binary[index] = '1';
        else
            binary[index] = '0';
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x, n;

    printf("Enter an integer (x): ");
    scanf("%d", &x);
    printf("Enter the number of positions to rotate (n): ");
    scanf("%d", &n);

    // Rotate Bits (Left)
    int rotate_left = (x << n) | (x >> (sizeof(x)*8 - n));
    printf("\nRotate %d (%s) to the left by %d positions: %d (%s)\n", x, decimalToBinary(x), n, rotate_left, decimalToBinary(rotate_left));

    // Rotate Bits (Right)
    int rotate_right = (x >> n) | (x << (sizeof(x)*8 - n));
    printf("Rotate %d (%s) to the right by %d positions: %d (%s)\n", x, decimalToBinary(x), n, rotate_right, decimalToBinary(rotate_right));

    return 0;
}

```

Output:

```
~ ./rotate_bits
Enter an integer (x): 25
Enter the number of positions to rotate (n): 5

Rotate 25 (00000000000000000000000000011001) to the left by 5 positions: 800 (00000000000000000000001100100000)
Rotate 25 (00000000000000000000000000011001) to the right by 5 positions: -939524096 (11001000000000000000000000000000)
```

13) Isolate Rightmost 1-Bit

int rightmost_set_bit = x & (-x);

14) Clear Rightmost 1-Bit

x = x & (x-1);

Program for the above:

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

char* decimalToBinary(int num) {
    char* binary = (char*)malloc(33);  // 32 bits + null terminator for an int
    binary[32] = '\0';
    int index = 31;

    for(int i = 0; i < 32; i++) {
        if((num & 1) == 1)
            binary[index] = '1';
        else
            binary[index] = '0';
        num >>= 1;
        index--;
    }
    return binary;
}

int main() {
    int x;

    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // Isolate Rightmost 1-Bit
    int rightmost_set_bit = x & (-x);
    printf("\nIsolate rightmost 1-bit of %d (%s): %d (%s)\n", x, decimalToBinary(x), rightmost_set_bit, decimalToBinary(rightmost_set_bit));

    // Clear Rightmost 1-Bit
    x = x & (x-1);
    printf("Clear rightmost 1-bit: %d (%s)\n", x, decimalToBinary(x));

    return 0;
}

```

Output:

```
~ ./right_most_bit
Enter an integer (x): 75

Isolate rightmost 1-bit of 75 (00000000000000000000000001001011): 1 (00000000000000000000000000000001)
Clear rightmost 1-bit: 74 (00000000000000000000000001001010)
```

15) Multiply by 2:

x = x << 1;

16) Divide by 2:

x = x >> 1;

Program for the above:

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

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;

    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // Multiply by 2
    int multiplyBy2 = x << 1;
    printf("\nMultiplying %d (%s) by 2: %d (%s)\n", x, decimalToBinaryWithSpace(x), multiplyBy2, decimalToBinaryWithSpace(multiplyBy2));

    // Divide by 2
    int divideBy2 = x >> 1;
    printf("Dividing %d (%s) by 2: %d (%s)\n", x, decimalToBinaryWithSpace(x), divideBy2, decimalToBinaryWithSpace(divideBy2));

    return 0;
}

```

Output:

```
~ ./by_2
Enter an integer (x): 379

Multiplying 379 ( 0000 0000 0000 0000 0001 0111 1011) by 2: 758 ( 0000 0000 0000 0000 0010 1111 0110)
Dividing 379 ( 0000 0000 0000 0000 0001 0111 1011) by 2: 189 ( 0000 0000 0000 0000 0000 1011 1101)
```

17) Flip all bits:
x = ~x;

18) Compute 2^x:
int result = 1 << x;

19) Check if a number is a power of 2:

bool isPowerOf2 = x && (!(x & (x - 1)));

Program for the above:

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

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;

    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // Flip all bits
    int flipped = ~x;
    printf("\nFlipping all bits of %d (%s): %d (%s)\n", x, decimalToBinaryWithSpace(x), flipped, decimalToBinaryWithSpace(flipped));

    // Compute 2^x
    int powerResult = 1 << x;
    printf("2^%d: %d (%s)\n", x, powerResult, decimalToBinaryWithSpace(powerResult));

    // Check if a number is a power of 2
    bool isPowerOf2 = x && (!(x & (x - 1)));
    printf("%d (%s) is %sa power of 2\n", x, decimalToBinaryWithSpace(x), isPowerOf2 ? "" : "not ");

    return 0;
}
```

Output:

```
~ ./flip_and_power_2
Enter an integer (x): 261

Flipping all bits of 261 ( 0000 0000 0000 0000 0001 0000 0101): -262 ( 1111 1111 1111 1111 1110 1111 1010)
2^261: 32 ( 0000 0000 0000 0000 0000 0010 0000)
261 ( 0000 0000 0000 0000 0001 0000 0101) is not a power of 2
```

20) Turn off all bits except the rightmost set bit:

x = x & (-x);

21) Turn on all bits to the right of rightmost set bit:

x = x | (x-1);

Program for the above:

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

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;

    printf("Enter an integer (x): ");
    scanf("%d", &x);

    // Turn off all bits except the rightmost set bit
    int rightmostSetBit = x & (-x);
    printf("\nTurn off all bits of %d (%s) except the rightmost set bit: %d (%s)\n", x, decimalToBinaryWithSpace(x), rightmostSetBit, decimalToBinaryWithSpace(rightmostSetBit));

    // Turn on all bits to the right of rightmost set bit
    int rightOfRightmost = x | (x-1);
    printf("Turn on all bits of %d (%s) to the right of rightmost set bit: %d (%s)\n", x, decimalToBinaryWithSpace(x), rightOfRightmost, decimalToBinaryWithSpace(rightOfRightmost));

    return 0;
}
```

Output:

```
~ ./bit_turning
Enter an integer (x): 3613

Turn off all bits of 3613 ( 0000 0000 0000 0000 1110 0001 1101) except the rightmost set bit: 1 ( 0000 0000 0000 0000 0000 0000 0001)
Turn on all bits of 3613 ( 0000 0000 0000 0000 1110 0001 1101) to the right of rightmost set bit: 3613 ( 0000 0000 0000 0000 1110 0001 1101)
~ vi bit_turning.c
```

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