Lecture 23

Bitwise Operation

RECURSION 23

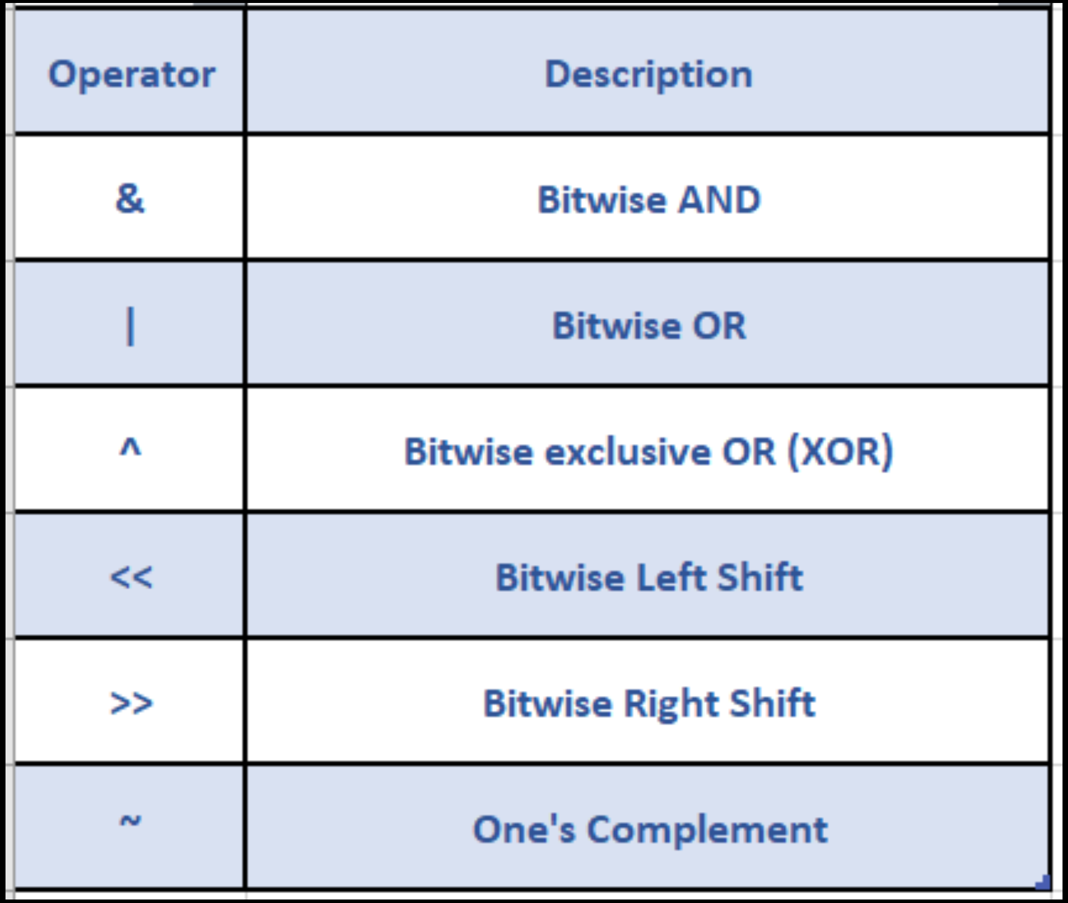
SIMPLIFIED CSE COURSE FOR ALL DEPARTMENTS

C & C++

REGISTER NOW

What are Bitwise Operators?

Operators that work on bits and perform bit-by-bit operations.
Commonly used in low-level programming.



Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (XOR)
<<	Bitwise Left Shift
>>	Bitwise Right Shift
~	One's Complement

Bit wise not

The expression for bitwise not is ~ (tilde)

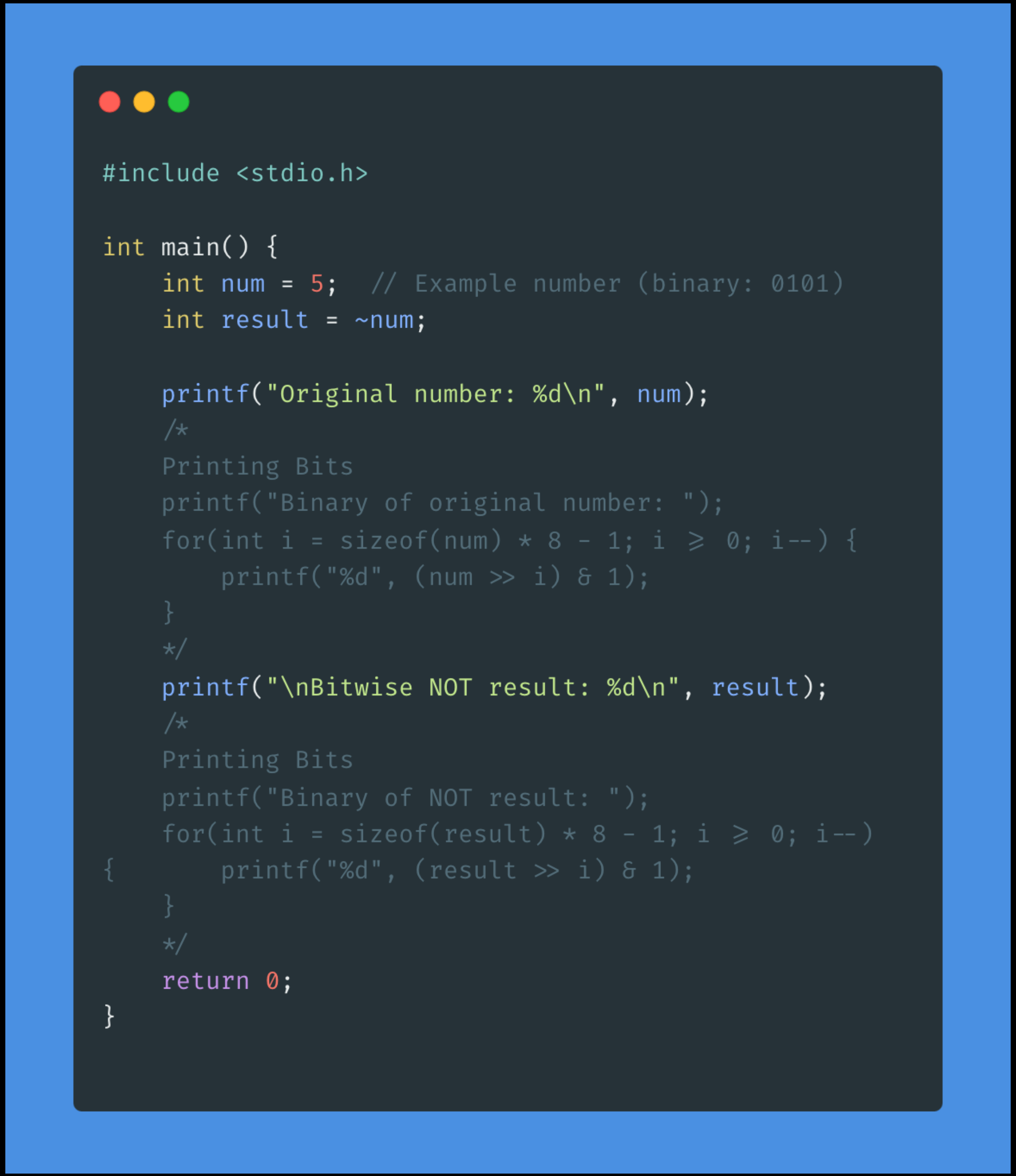
It is a unary operator.

Flips all the bits of a number.

Example:

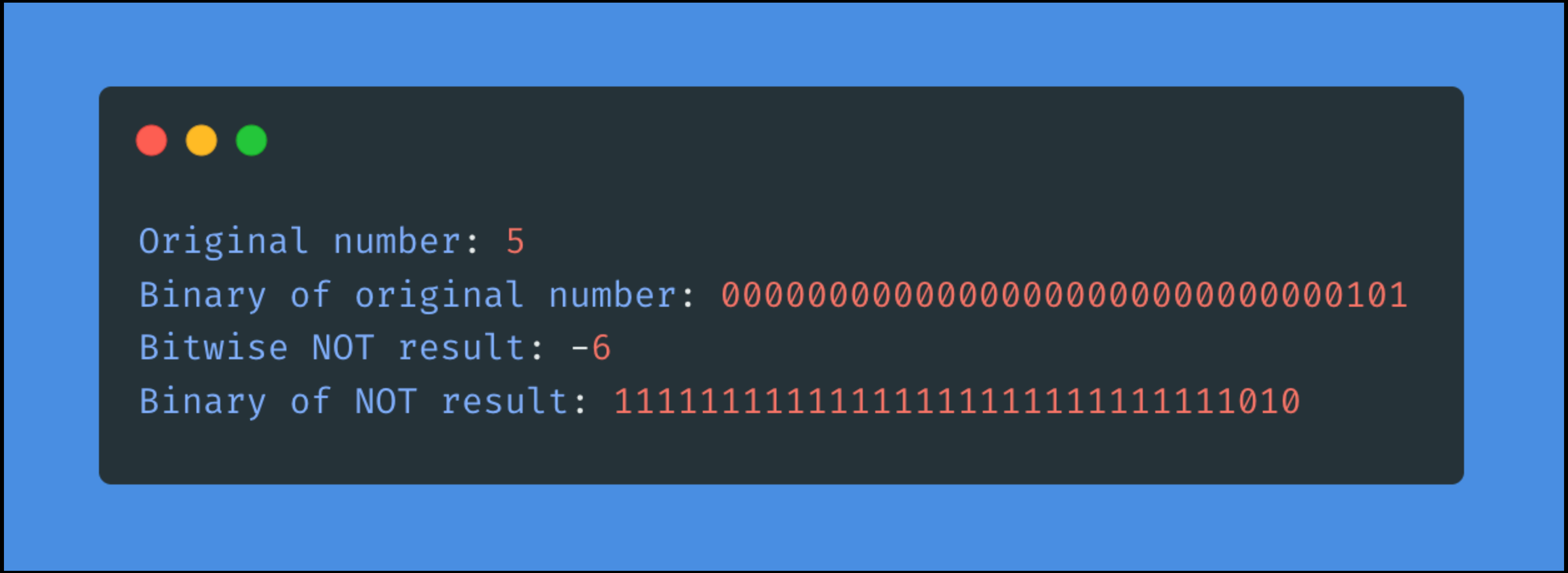
5 = 0000101
~5 = 1111010

Example



```c
#include <stdio.h>

int main() {
    int num = 5;  // Example number (binary: 0101)
    int result = ~num;

    printf("Original number: %d\n", num);
    /*
    Printing Bits
    printf("Binary of original number: ");
    for(int i = sizeof(num) * 8 - 1; i >= 0; i--) {
        printf("%d", (num >> i) & 1);
    }
    */
    printf("\nBitwise NOT result: %d\n", result);
    /*
    Printing Bits
    printf("Binary of NOT result: ");
    for(int i = sizeof(result) * 8 - 1; i >= 0; i--)
{       printf("%d", (result >> i) & 1);
    }
    */
    return 0;
}
```

Output



```
Original number: 5
Binary of original number: 00000000000000000000000000000101
Bitwise NOT result: -6
Binary of NOT result: 11111111111111111111111111111010
```

2’s Complement

1. Negative Numbers Representation in Computers:
   - In computers, negative numbers are represented using a method called 2's complement.

2. Bitwise Operation to Obtain 2's Complement:
   - To find the 2's complement of a binary number:
     1. Start from the rightmost bit and leave all bits unchanged until you encounter the first `1`.
     2. Flip all the bits to the left of this `1`.

3. Why Use 2's Complement?
   - Simplifies Arithmetic Operations: Using 2's complement simplifies the design of arithmetic circuits in computers, as both addition and subtraction can be performed using the same hardware.
   - Unique Zero Representation: There is only one representation for zero, unlike sign-and-magnitude and one's complement methods which have two representations for zero.

4. Example of Converting a Positive Number to 2's Complement:
   - Let's take the positive number `5` and find its 2's complement to represent `-5`:
     1. Binary Representation of 5: `00000101`
     2. Invert All Bits: `11111010`
     3. Add 1 to the Inverted Bits: `11111010 + 1 = 11111011`
   - Therefore, `-5` in 2's complement representation is `11111011`.

Shift Operators

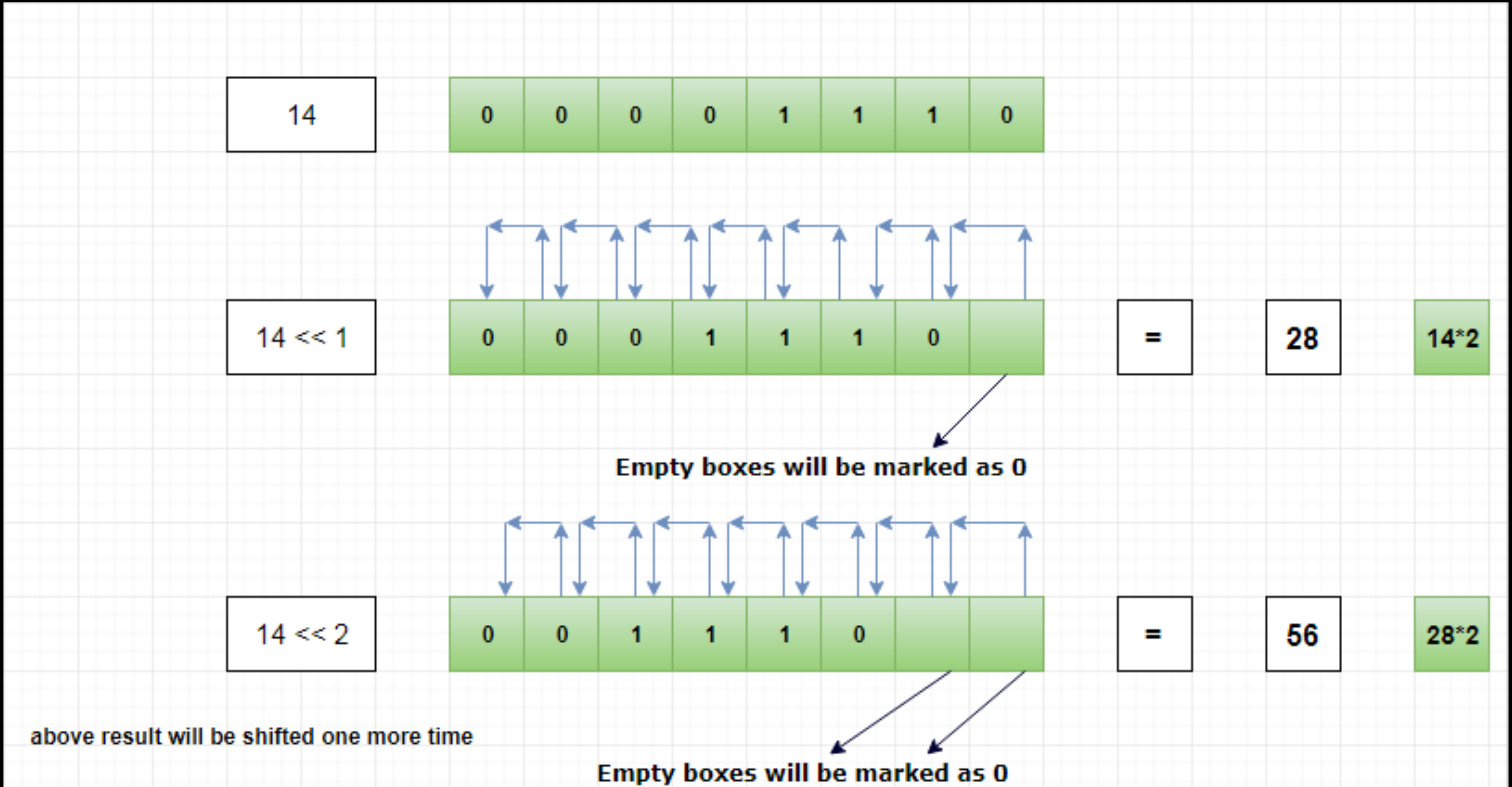
Left Shift (<<):

- Shifts bits to the left.
- Adds zeros to the right.
- Discards bits shifted out on the left.
- Example: `number << n`

Right Shift (>>):

- Shifts bits to the right.
- Adds zeros (for unsigned) or the sign bit (for signed) to the left.
- Discards bits shifted out on the right.
- Example: `number >> n`

Left Shift



14	0	0	0	0	1	1	1	0			
14 << 1	0	0	0	1	1	1	0		=	28	14*2
14 << 2	0	0	1	1	1	0			=	56	28*2

Empty boxes will be marked as 0

above result will be shifted one more time

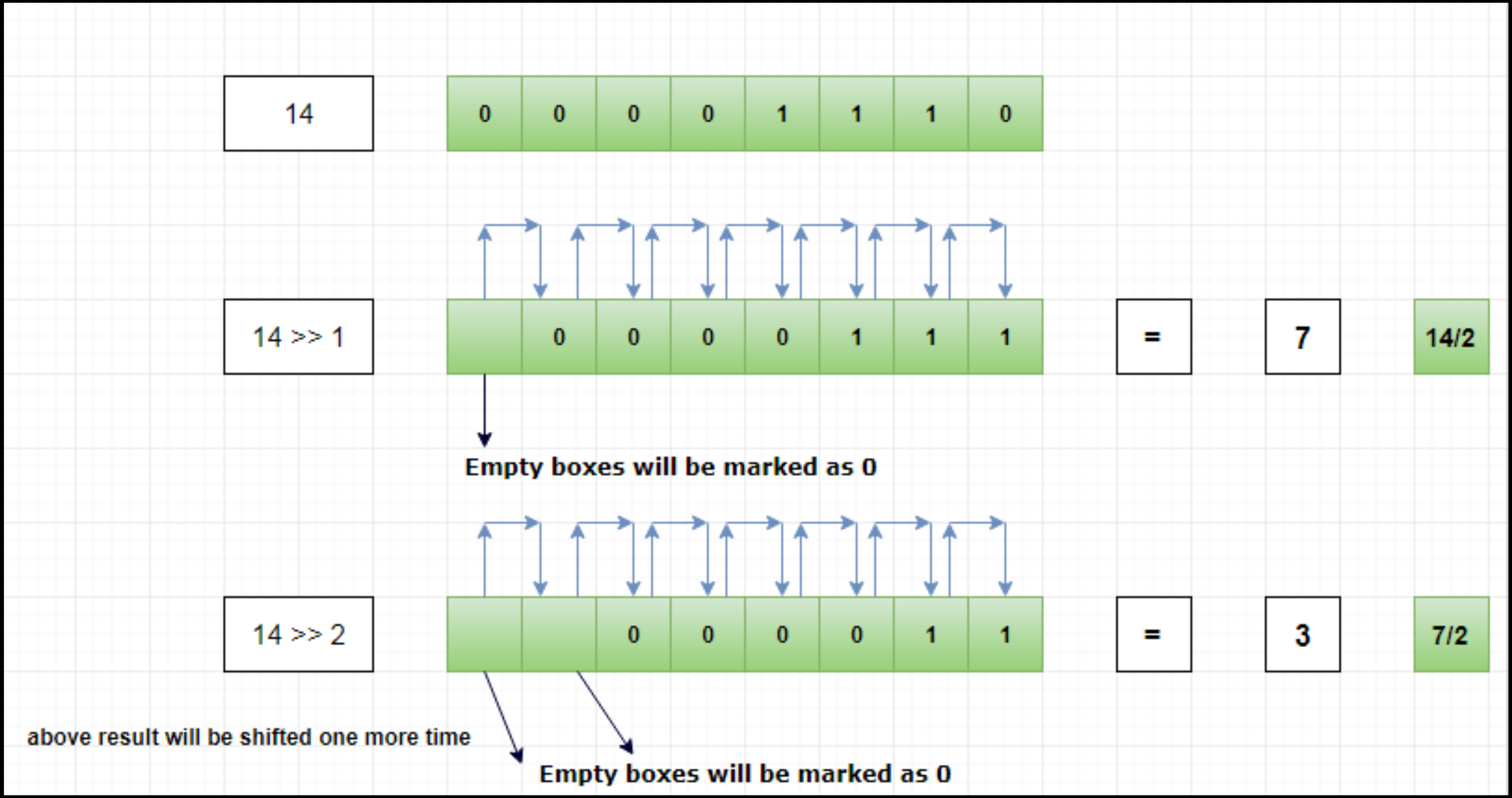
Empty boxes will be marked as 0

Formula

Left Shift (<<): `number << n = number*(2^n)`

- Example:
- 10<<4 = 10*(2^4) = 160

Right Shift



14	0	0	0	0	1	1	1	0			
14 >> 1		0	0	0	0	1	1	1	=	7	14/2
14 >> 2			0	0	0	0	1	1	=	3	7/2

Empty boxes will be marked as 0

above result will be shifted one more time

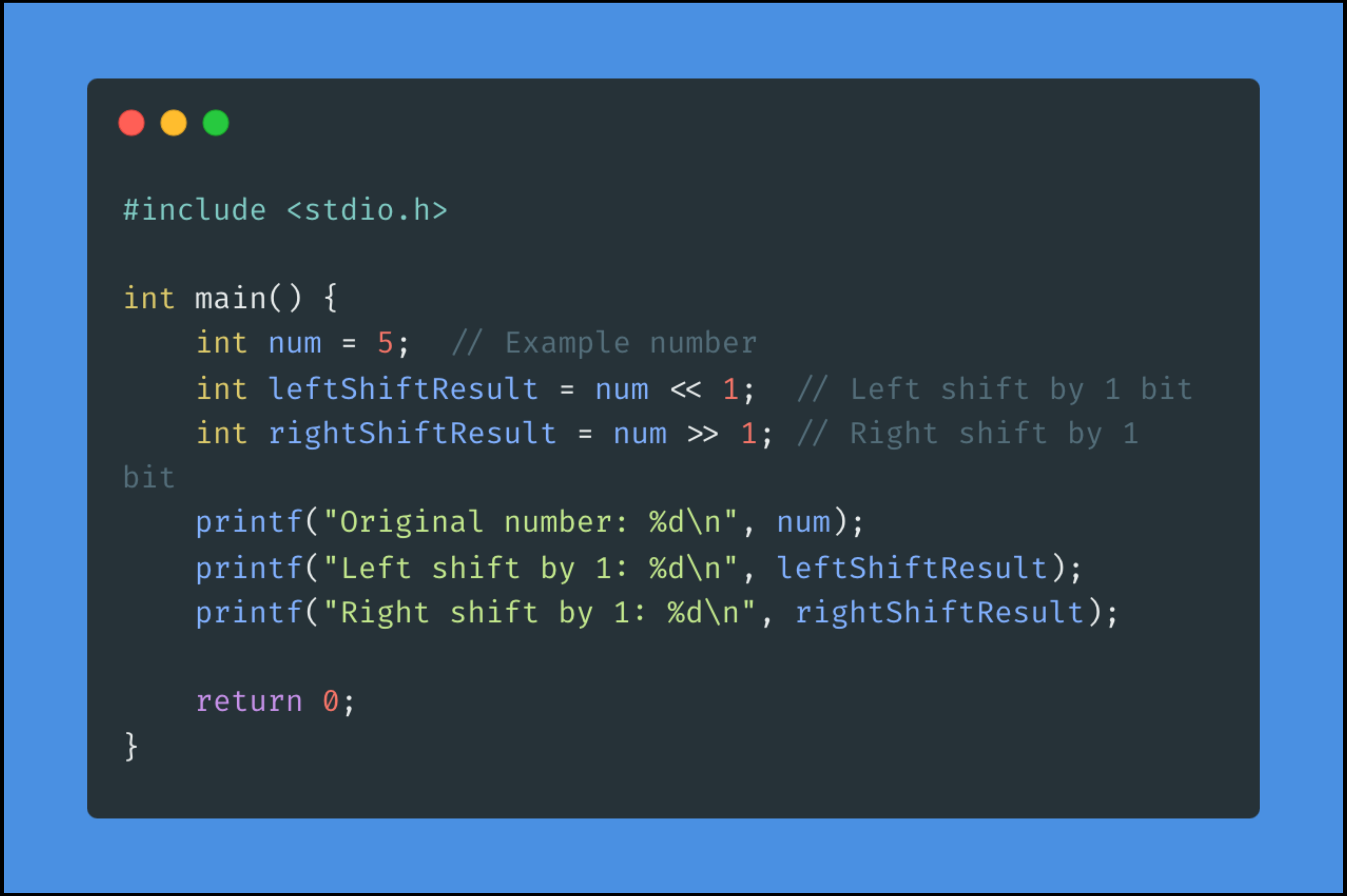
Empty boxes will be marked as 0

Formula

Left Shift (<<): `number << n = number/(2^n)`

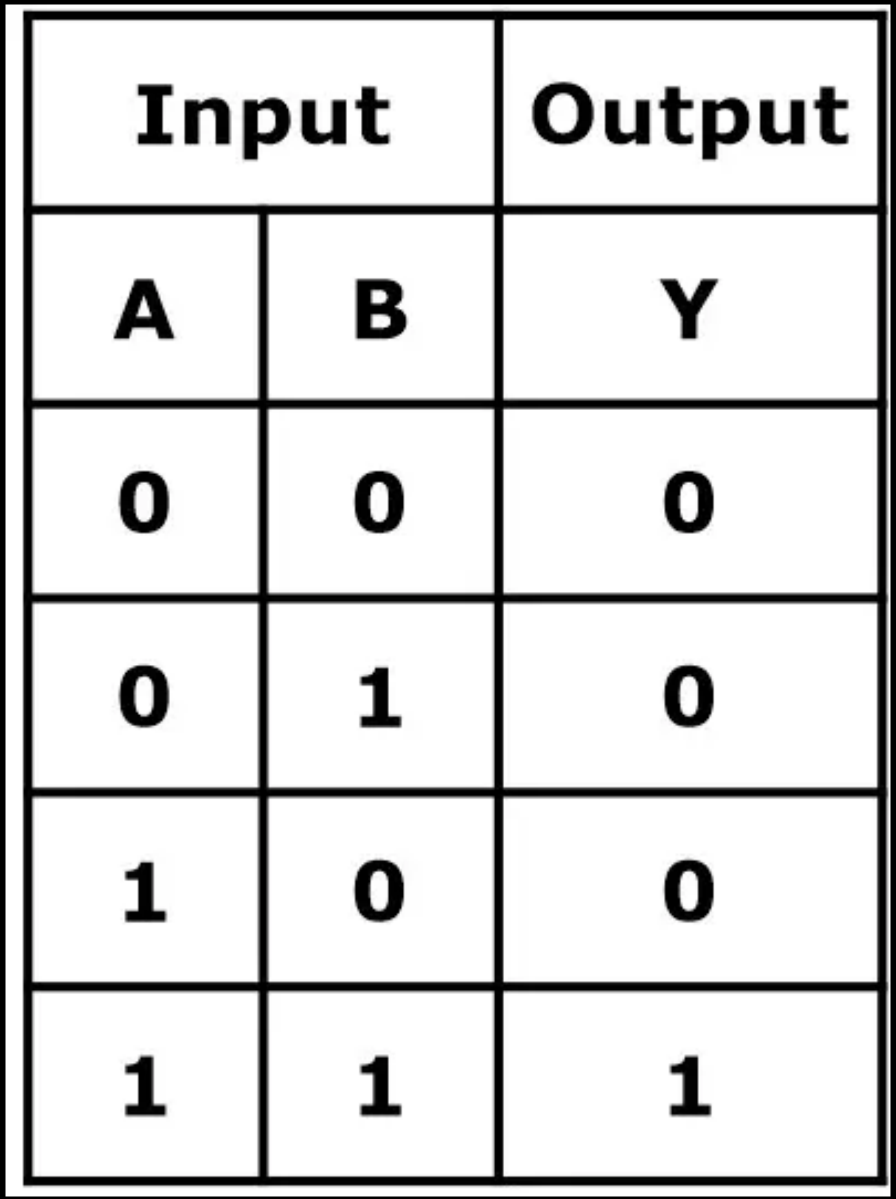
- Example:
- 10<<4 = 10/(2^4) = 10/16 = 0

Shift Operator Example



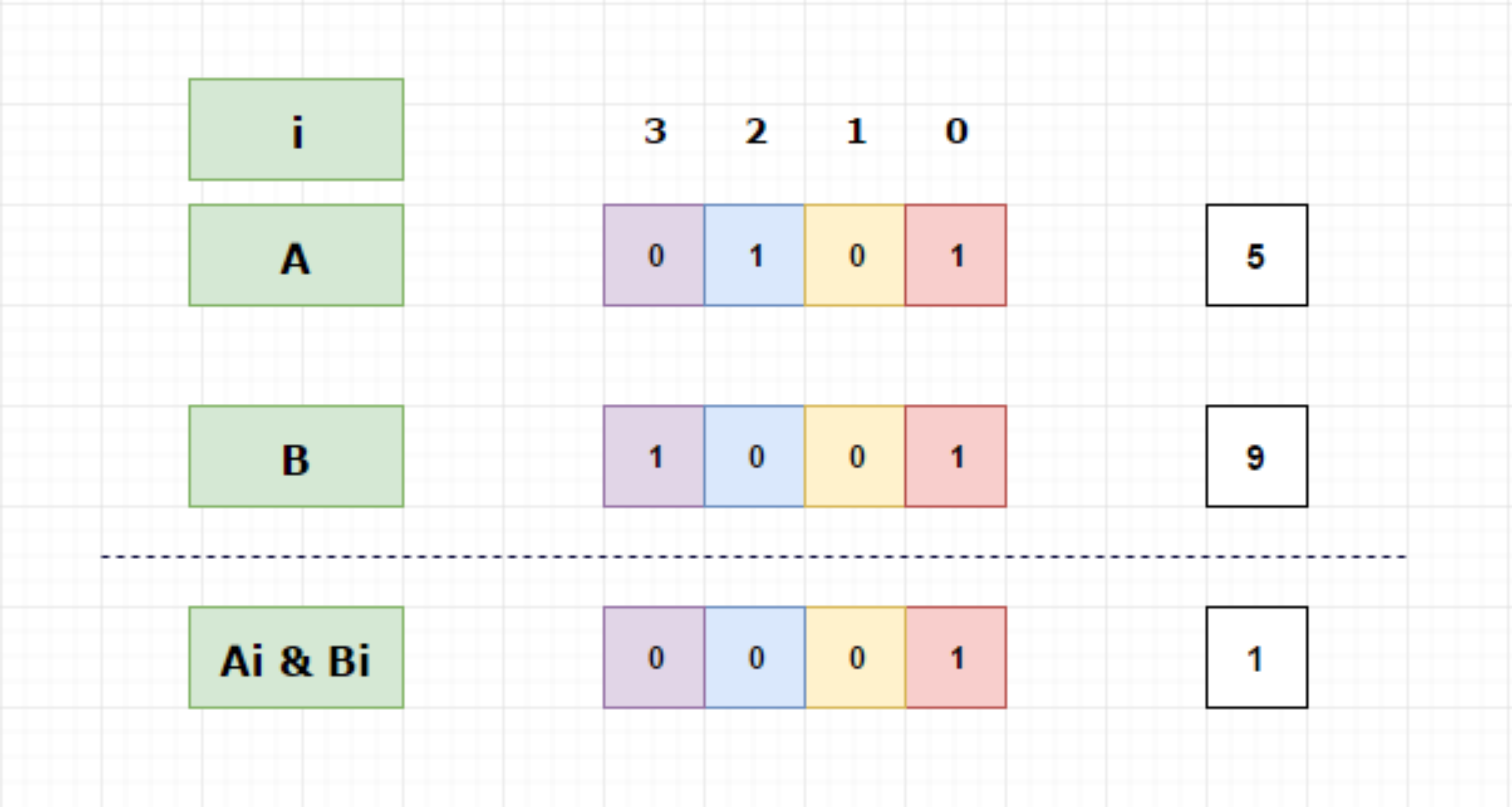
```c
#include <stdio.h>

int main() {
    int num = 5;  // Example number
    int leftShiftResult = num << 1;  // Left shift by 1 bit
    int rightShiftResult = num >> 1; // Right shift by 1
bit
    printf("Original number: %d\n", num);
    printf("Left shift by 1: %d\n", leftShiftResult);
    printf("Right shift by 1: %d\n", rightShiftResult);

    return 0;
}
```

And Truth Table



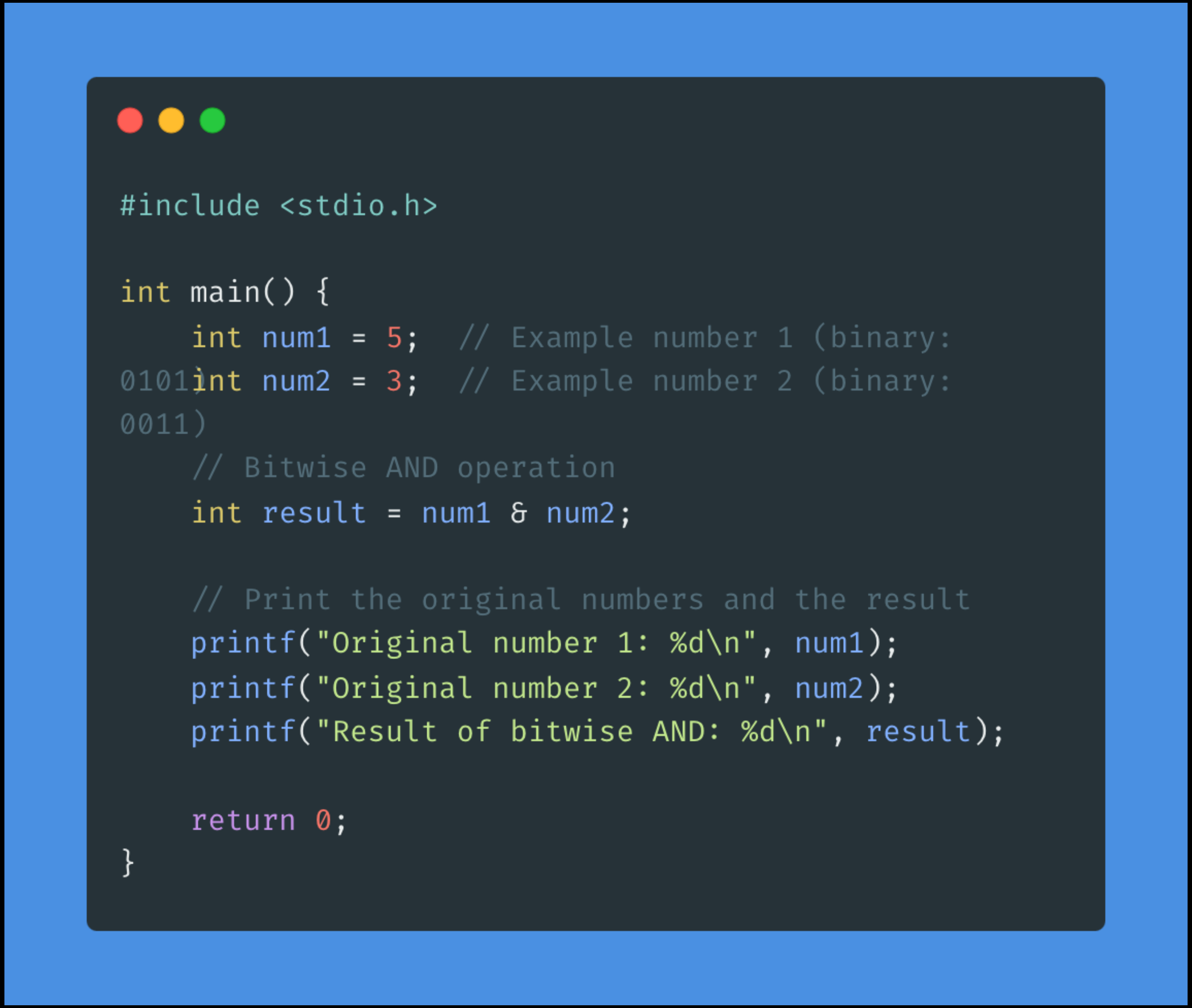
Input		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND Operator Example



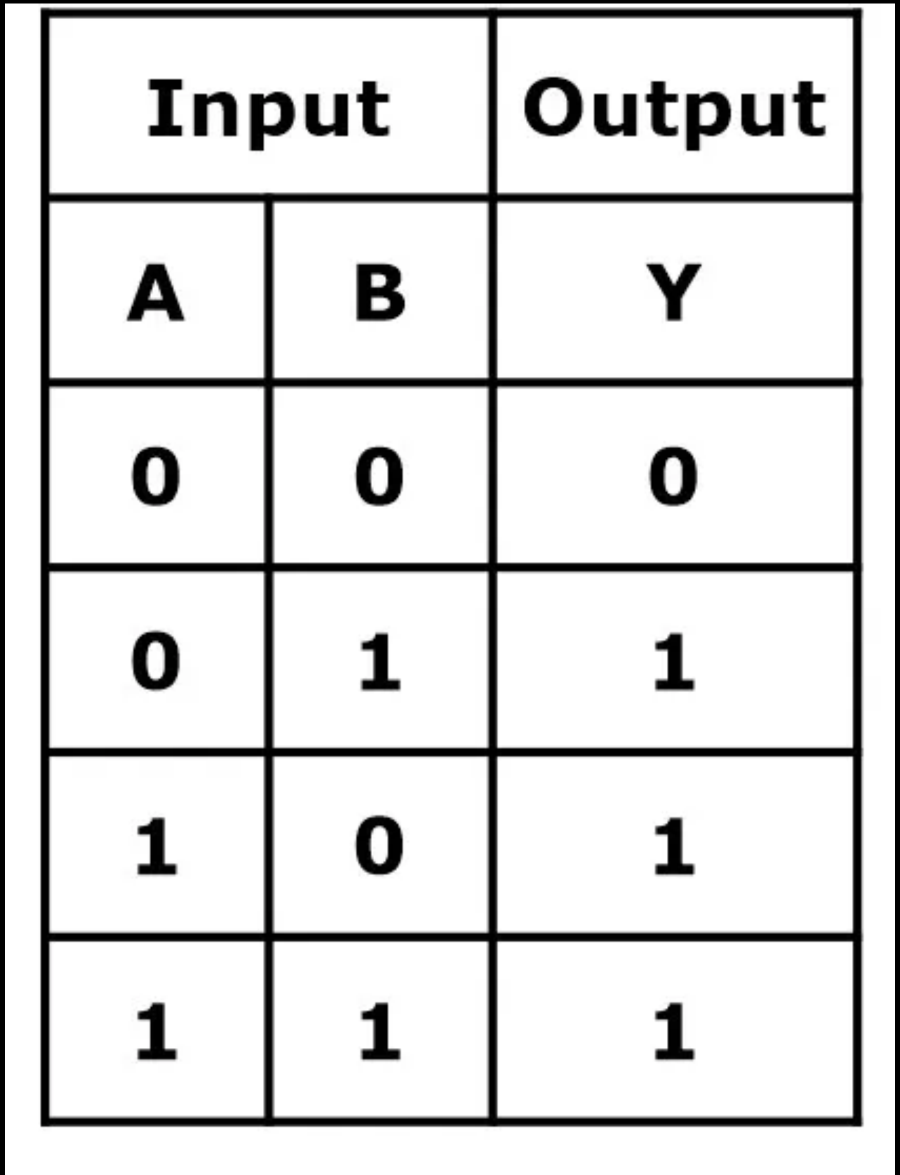
i	3	2	1	0	
A	0	1	0	1	5
B	1	0	0	1	9
Ai & Bi	0	0	0	1	1

AND Operator Code



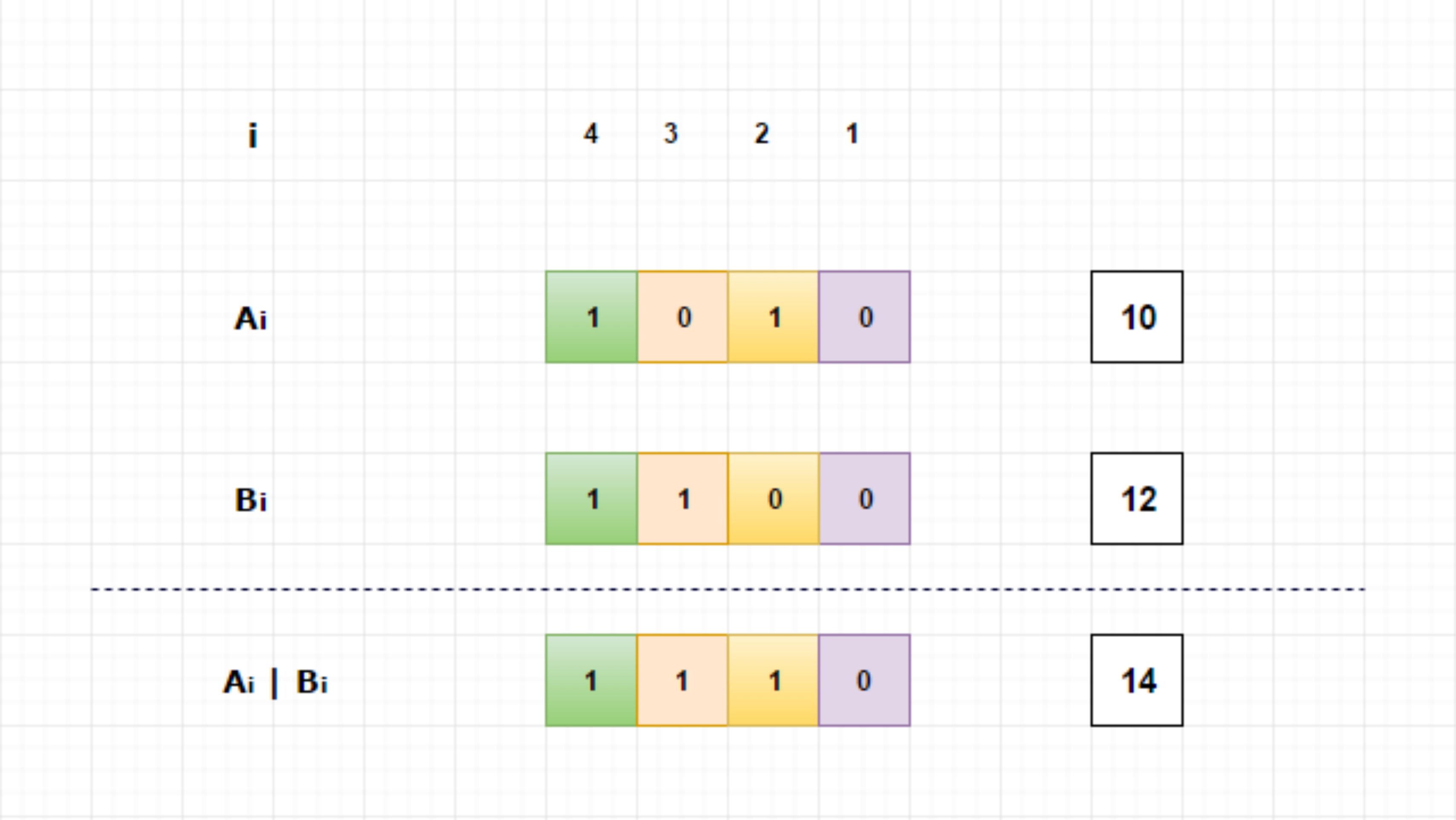
```
#include <stdio.h>

int main() {
    int num1 = 5;  // Example number 1 (binary:
0101int num2 = 3;  // Example number 2 (binary:
0011)
    // Bitwise AND operation
    int result = num1 & num2;

    // Print the original numbers and the result
    printf("Original number 1: %d\n", num1);
    printf("Original number 2: %d\n", num2);
    printf("Result of bitwise AND: %d\n", result);

    return 0;
}
```

OR Truth Table



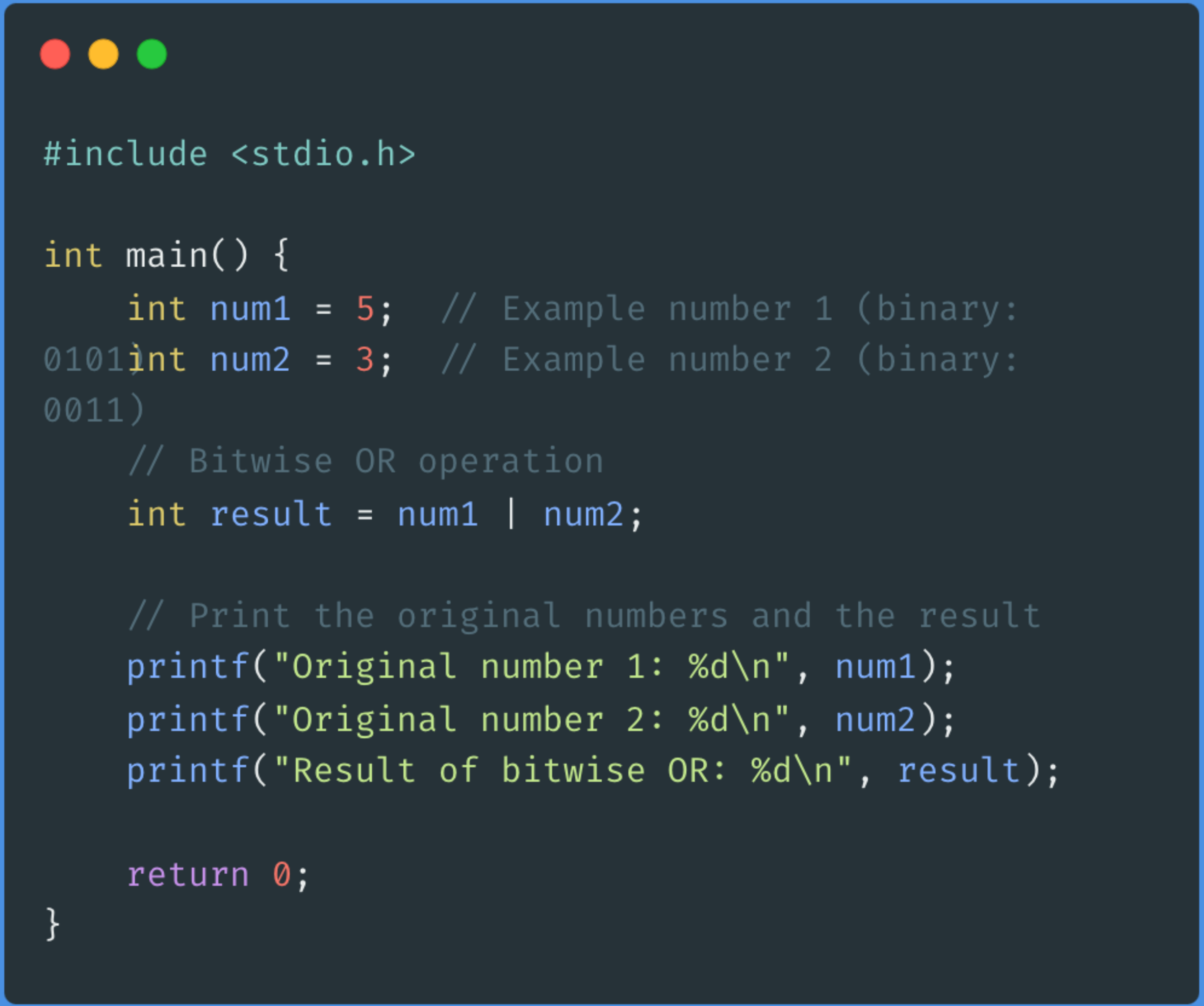
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

OR Example



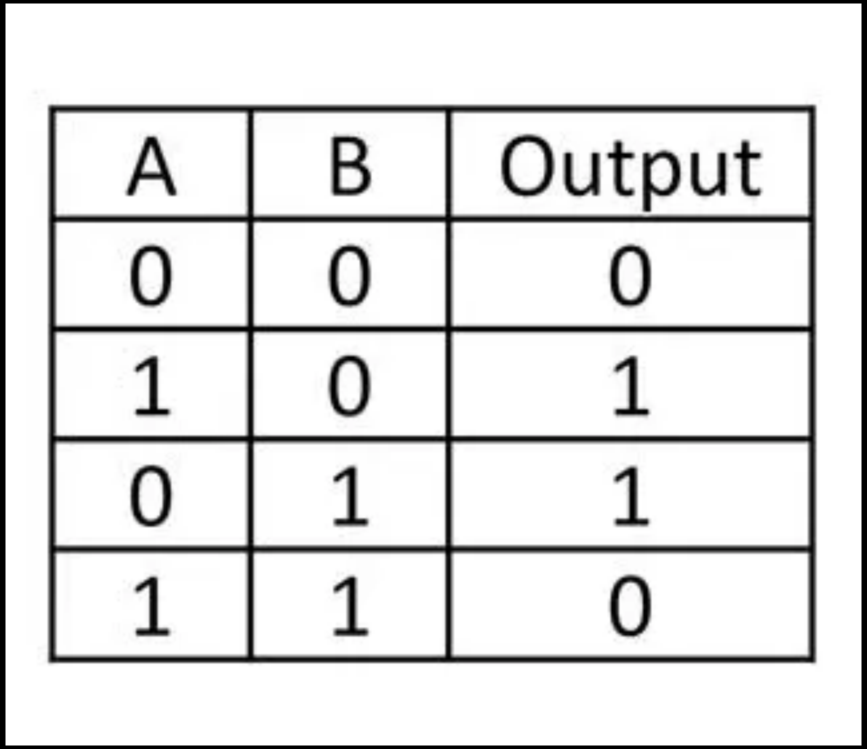
i	4	3	2	1	
A_i	1	0	1	0	10
B_i	1	1	0	0	12
A_i | B_i	1	1	1	0	14

OR Code



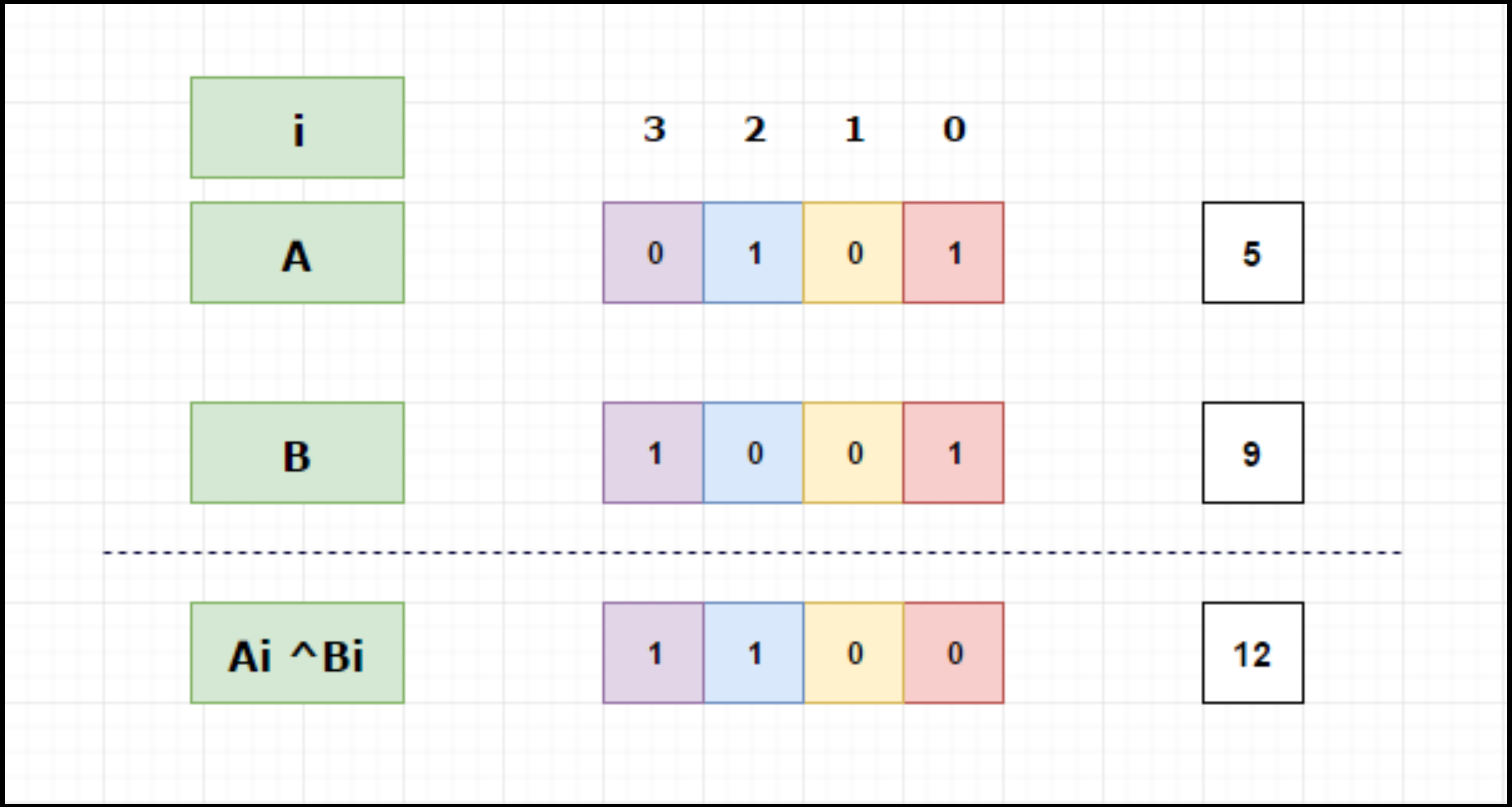
```
#include <stdio.h>

int main() {
    int num1 = 5;  // Example number 1 (binary:
0101)int num2 = 3;  // Example number 2 (binary:
0011)
    // Bitwise OR operation
    int result = num1 | num2;

    // Print the original numbers and the result
    printf("Original number 1: %d\n", num1);
    printf("Original number 2: %d\n", num2);
    printf("Result of bitwise OR: %d\n", result);

    return 0;
}
```

XOR Truth Table



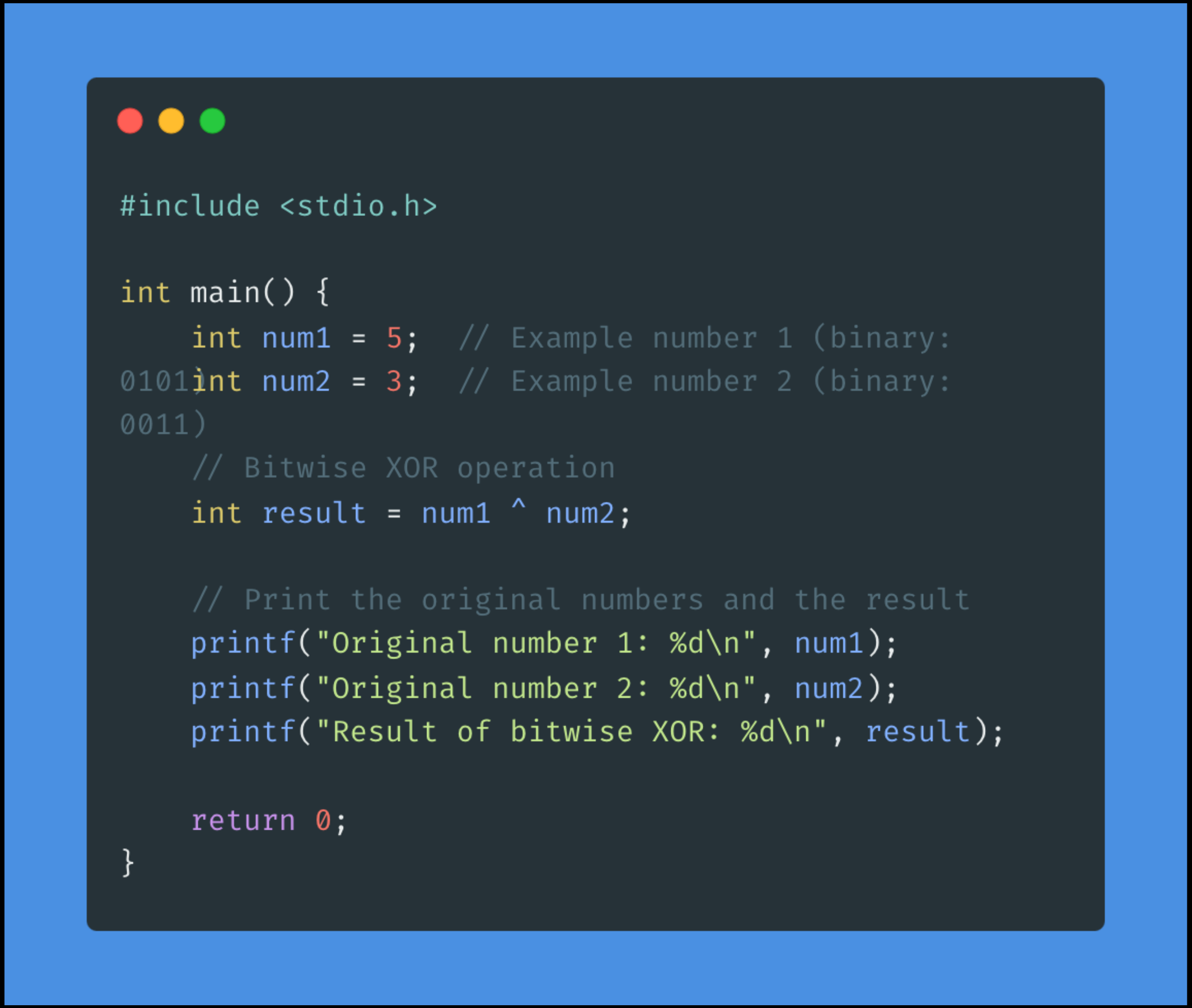
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

XOR Example



i	3	2	1	0	
A	0	1	0	1	5
B	1	0	0	1	9
Ai ^Bi	1	1	0	0	12

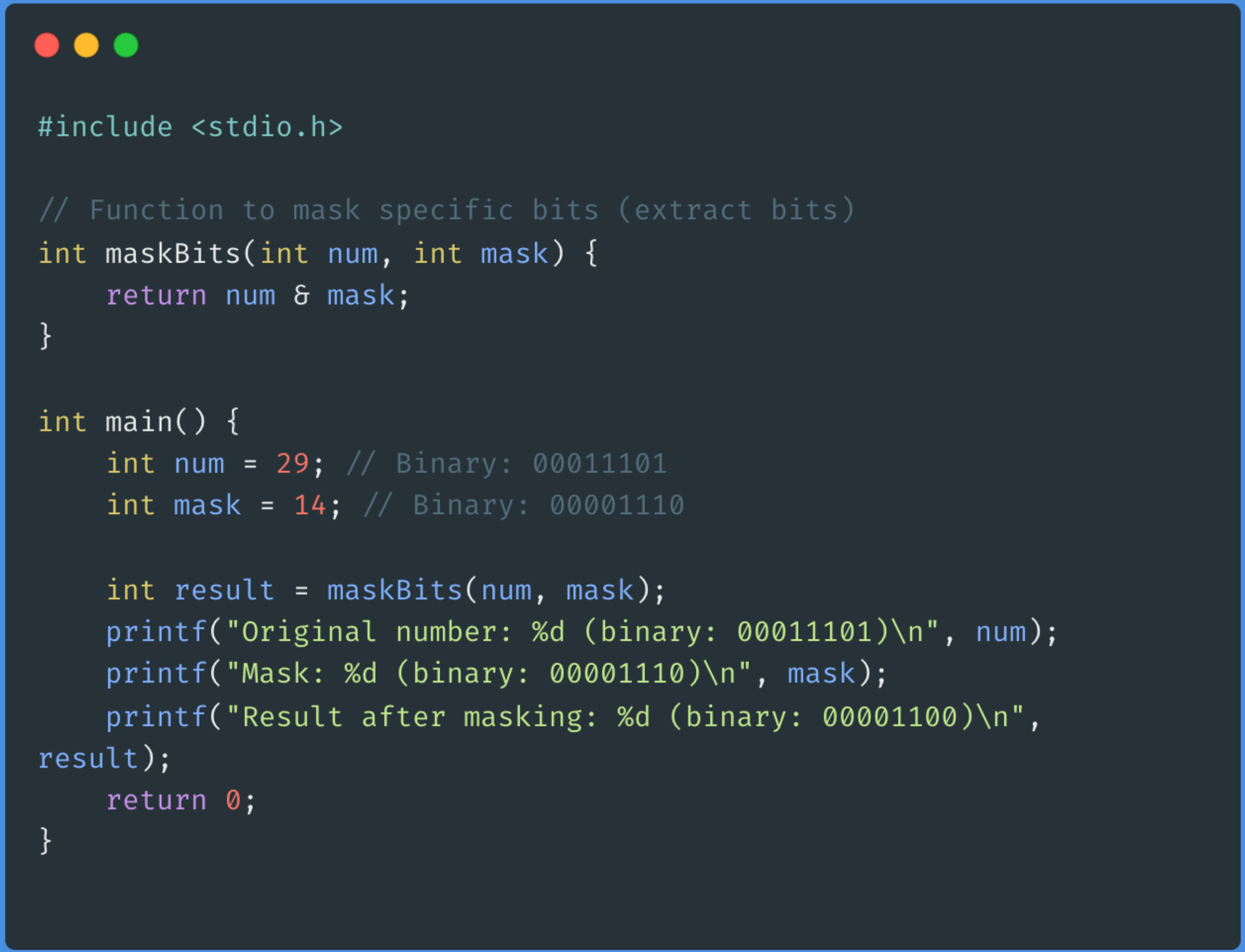
XOR Code



```
#include <stdio.h>

int main() {
    int num1 = 5;   // Example number 1 (binary:
0101int num2 = 3;   // Example number 2 (binary:
0011)
    // Bitwise XOR operation
    int result = num1 ^ num2;

    // Print the original numbers and the result
    printf("Original number 1: %d\n", num1);
    printf("Original number 2: %d\n", num2);
    printf("Result of bitwise XOR: %d\n", result);

    return 0;
}
```

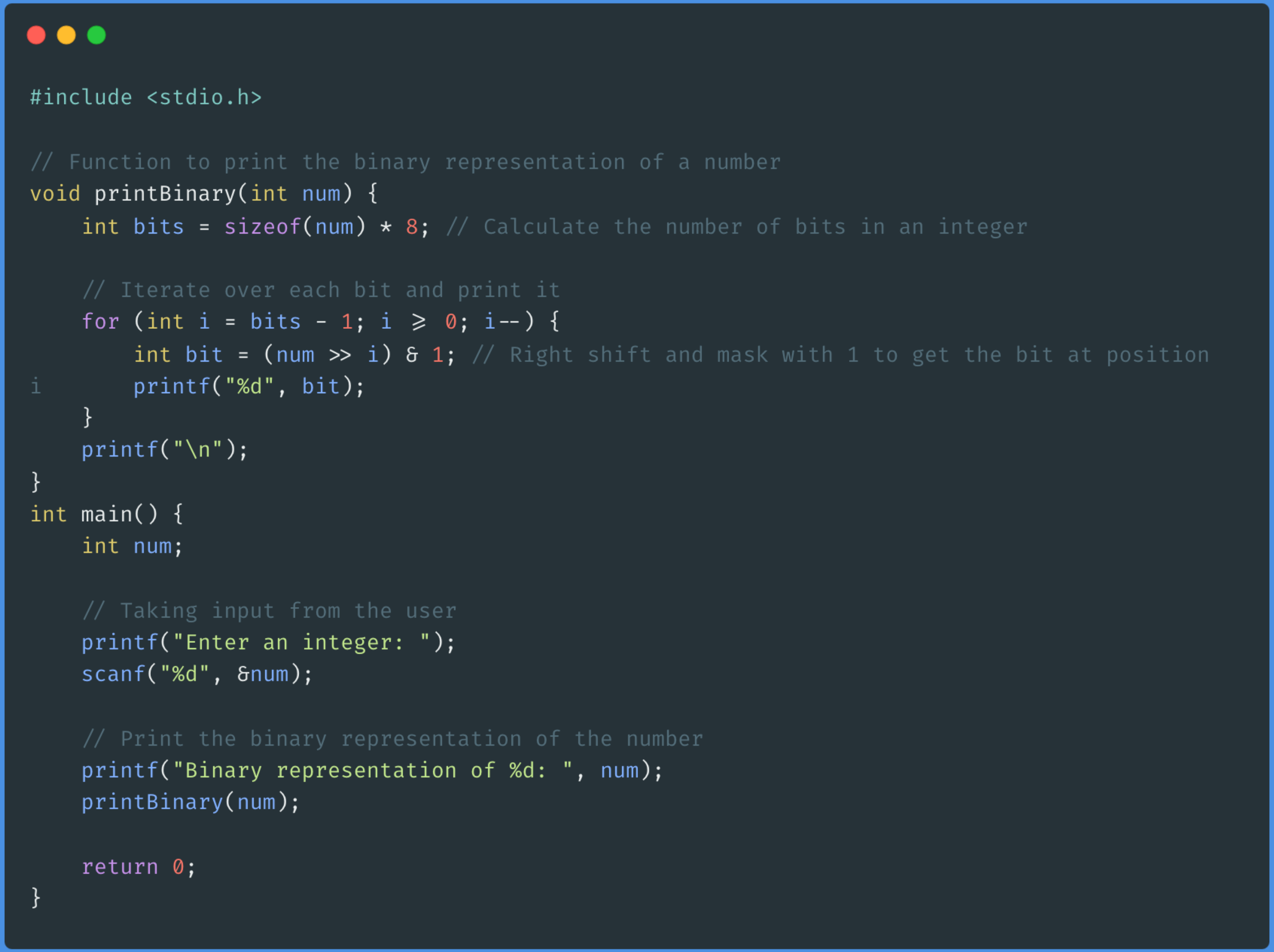
Bit Manipulation

Masking Bits



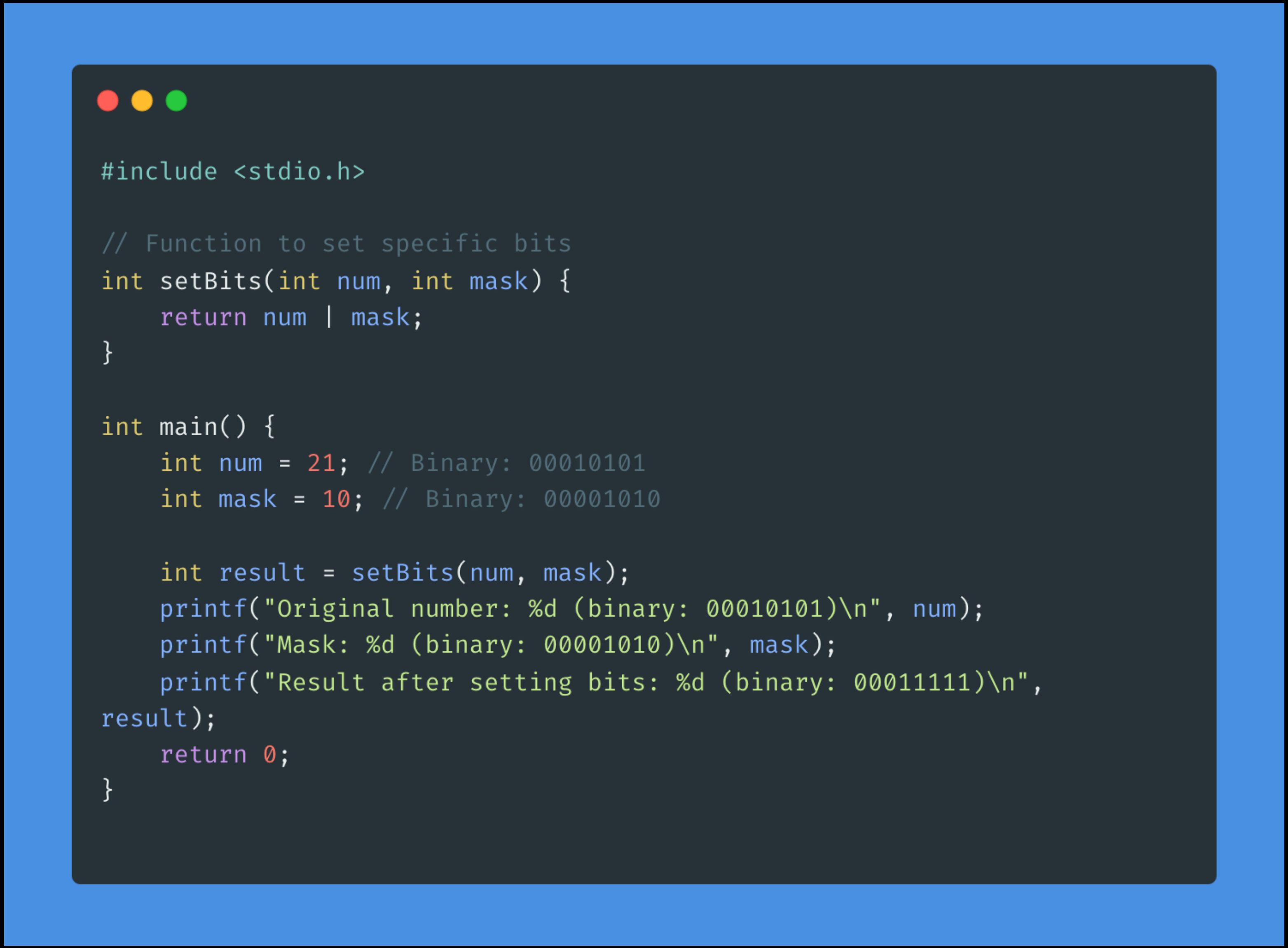
```c
#include <stdio.h>

// Function to mask specific bits (extract bits)
int maskBits(int num, int mask) {
    return num & mask;
}

int main() {
    int num = 29; // Binary: 00011101
    int mask = 14; // Binary: 00001110

    int result = maskBits(num, mask);
    printf("Original number: %d (binary: 00011101)\n", num);
    printf("Mask: %d (binary: 00001110)\n", mask);
    printf("Result after masking: %d (binary: 00001100)\n",
result);
    return 0;
}
```

Printing Bits of Number



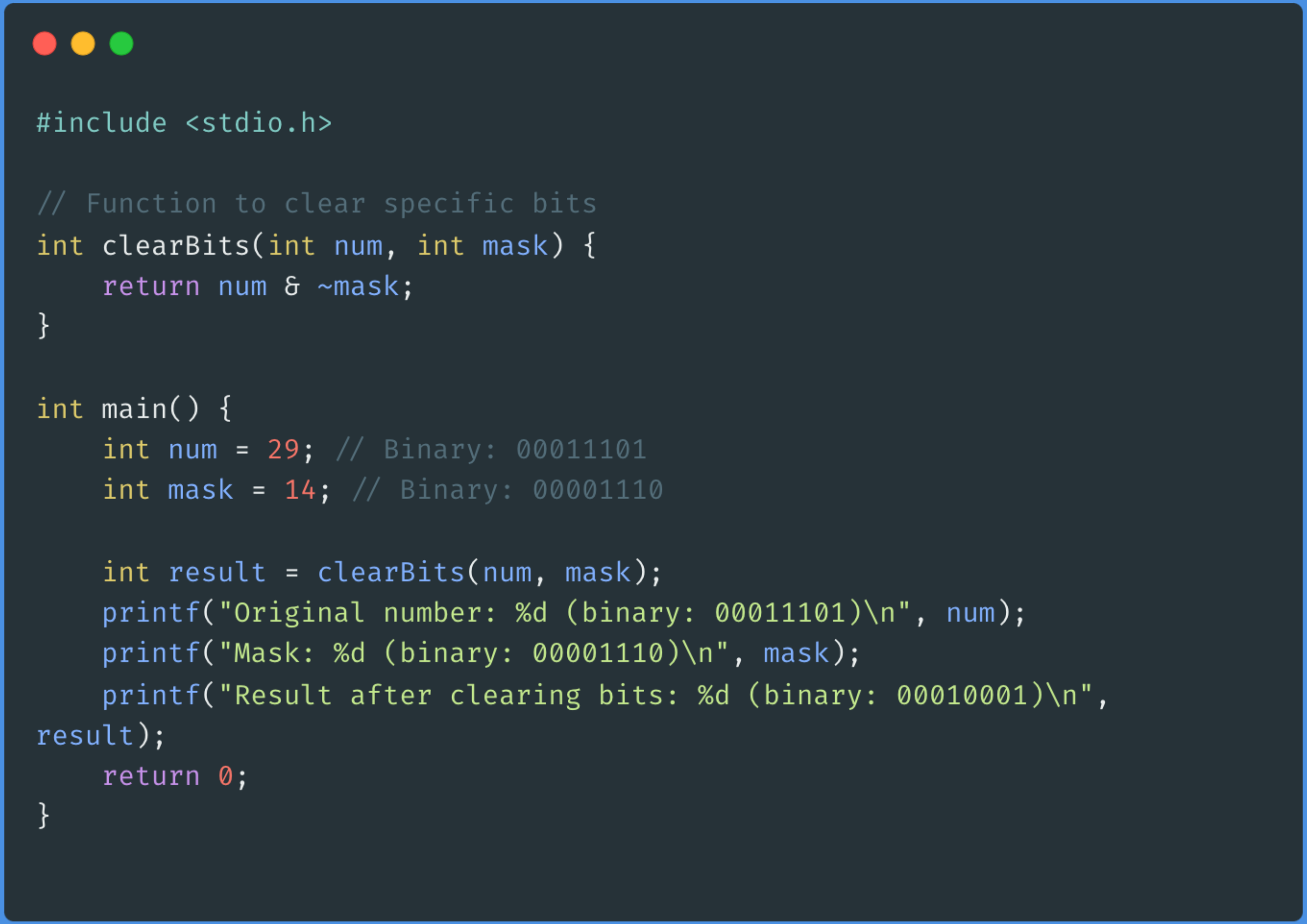
```c
#include <stdio.h>

// Function to print the binary representation of a number
void printBinary(int num) {
    int bits = sizeof(num) * 8; // Calculate the number of bits in an integer

    // Iterate over each bit and print it
    for (int i = bits - 1; i >= 0; i--) {
        int bit = (num >> i) & 1; // Right shift and mask with 1 to get the bit at position
i       printf("%d", bit);
    }
    printf("\n");
}
int main() {
    int num;

    // Taking input from the user
    printf("Enter an integer: ");
    scanf("%d", &num);

    // Print the binary representation of the number
    printf("Binary representation of %d: ", num);
    printBinary(num);

    return 0;
}
```

Setting Bits



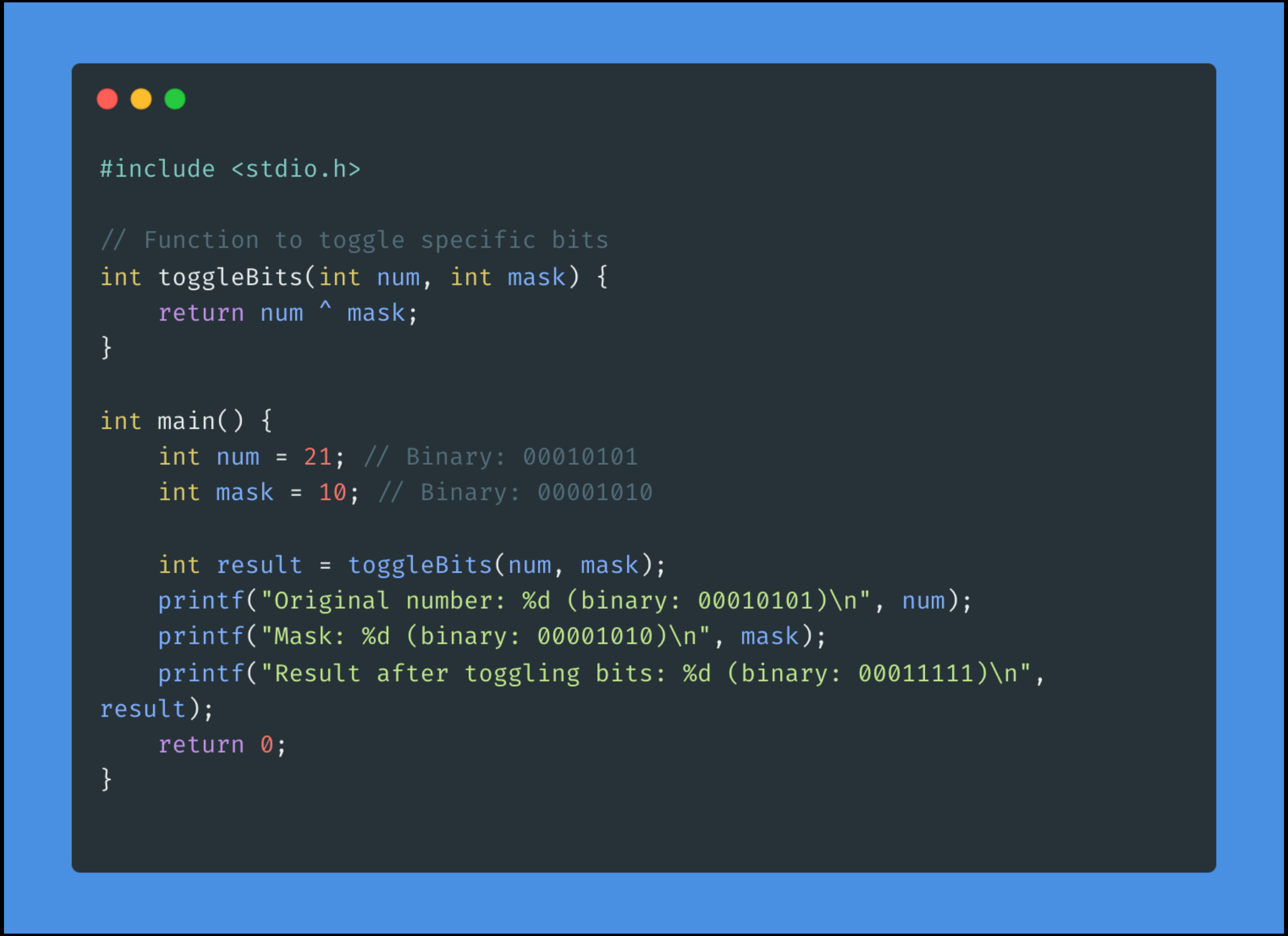
```c
#include <stdio.h>

// Function to set specific bits
int setBits(int num, int mask) {
    return num | mask;
}

int main() {
    int num = 21; // Binary: 00010101
    int mask = 10; // Binary: 00001010

    int result = setBits(num, mask);
    printf("Original number: %d (binary: 00010101)\n", num);
    printf("Mask: %d (binary: 00001010)\n", mask);
    printf("Result after setting bits: %d (binary: 00011111)\n",
result);
    return 0;
}
```

Clearing Bits



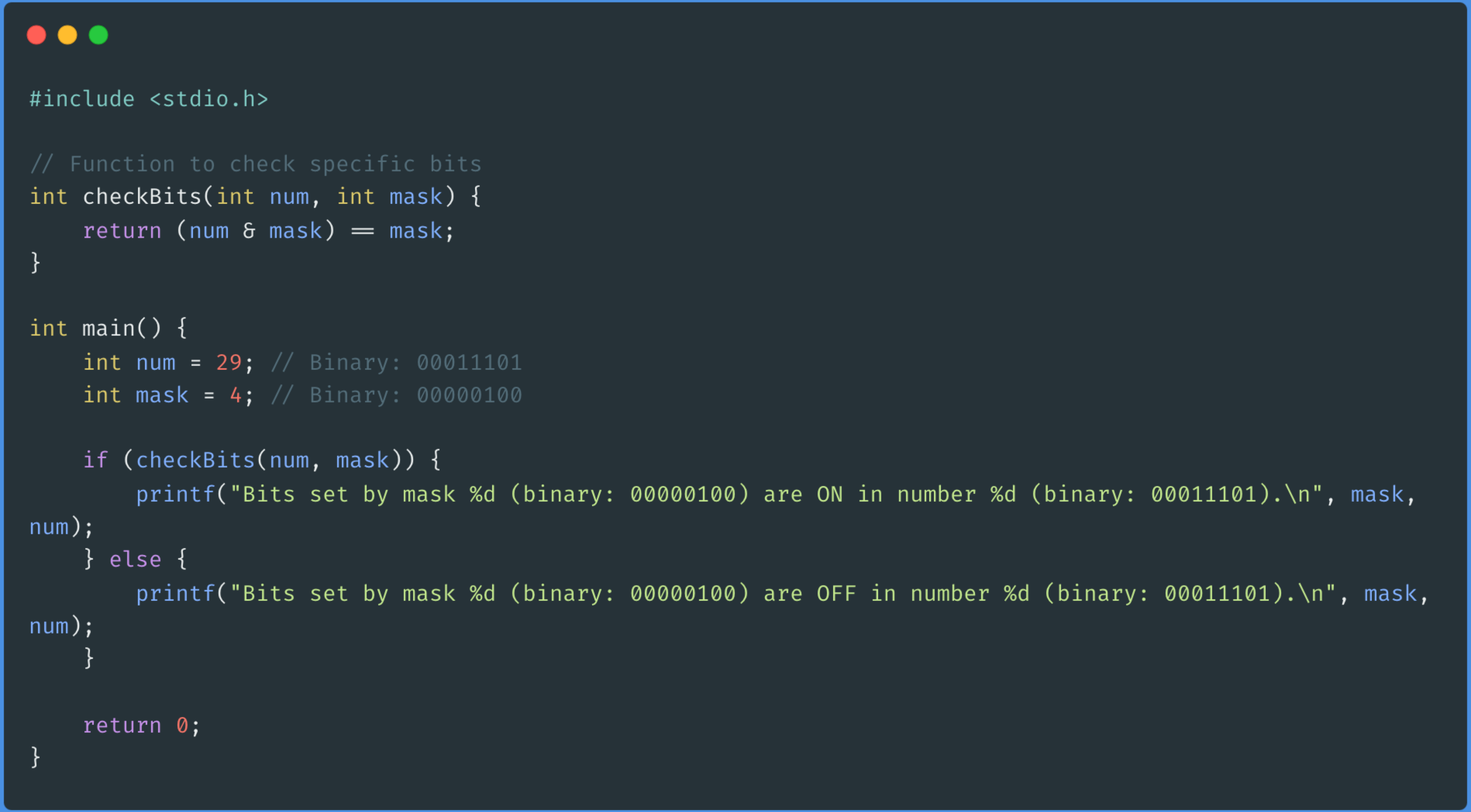
```c
#include <stdio.h>

// Function to clear specific bits
int clearBits(int num, int mask) {
    return num & ~mask;
}

int main() {
    int num = 29; // Binary: 00011101
    int mask = 14; // Binary: 00001110

    int result = clearBits(num, mask);
    printf("Original number: %d (binary: 00011101)\n", num);
    printf("Mask: %d (binary: 00001110)\n", mask);
    printf("Result after clearing bits: %d (binary: 00010001)\n",
result);
    return 0;
}
```

Toggling Bits



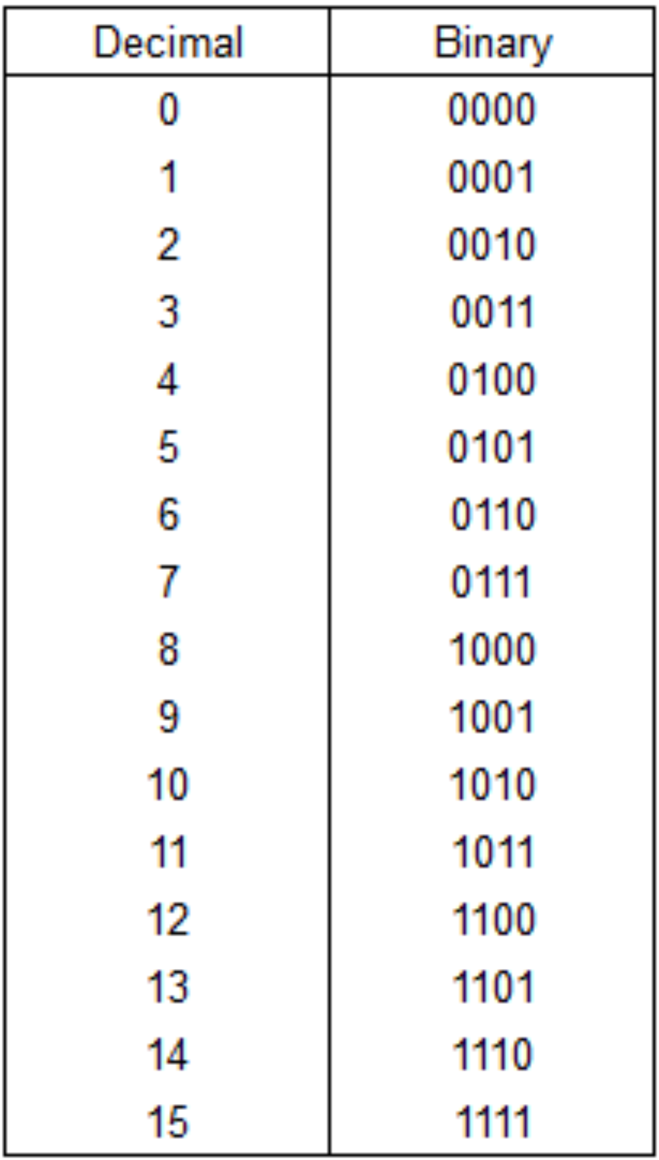
```c
#include <stdio.h>

// Function to toggle specific bits
int toggleBits(int num, int mask) {
    return num ^ mask;
}

int main() {
    int num = 21; // Binary: 00010101
    int mask = 10; // Binary: 00001010

    int result = toggleBits(num, mask);
    printf("Original number: %d (binary: 00010101)\n", num);
    printf("Mask: %d (binary: 00001010)\n", mask);
    printf("Result after toggling bits: %d (binary: 00011111)\n",
result);
    return 0;
}
```

Checking Bits



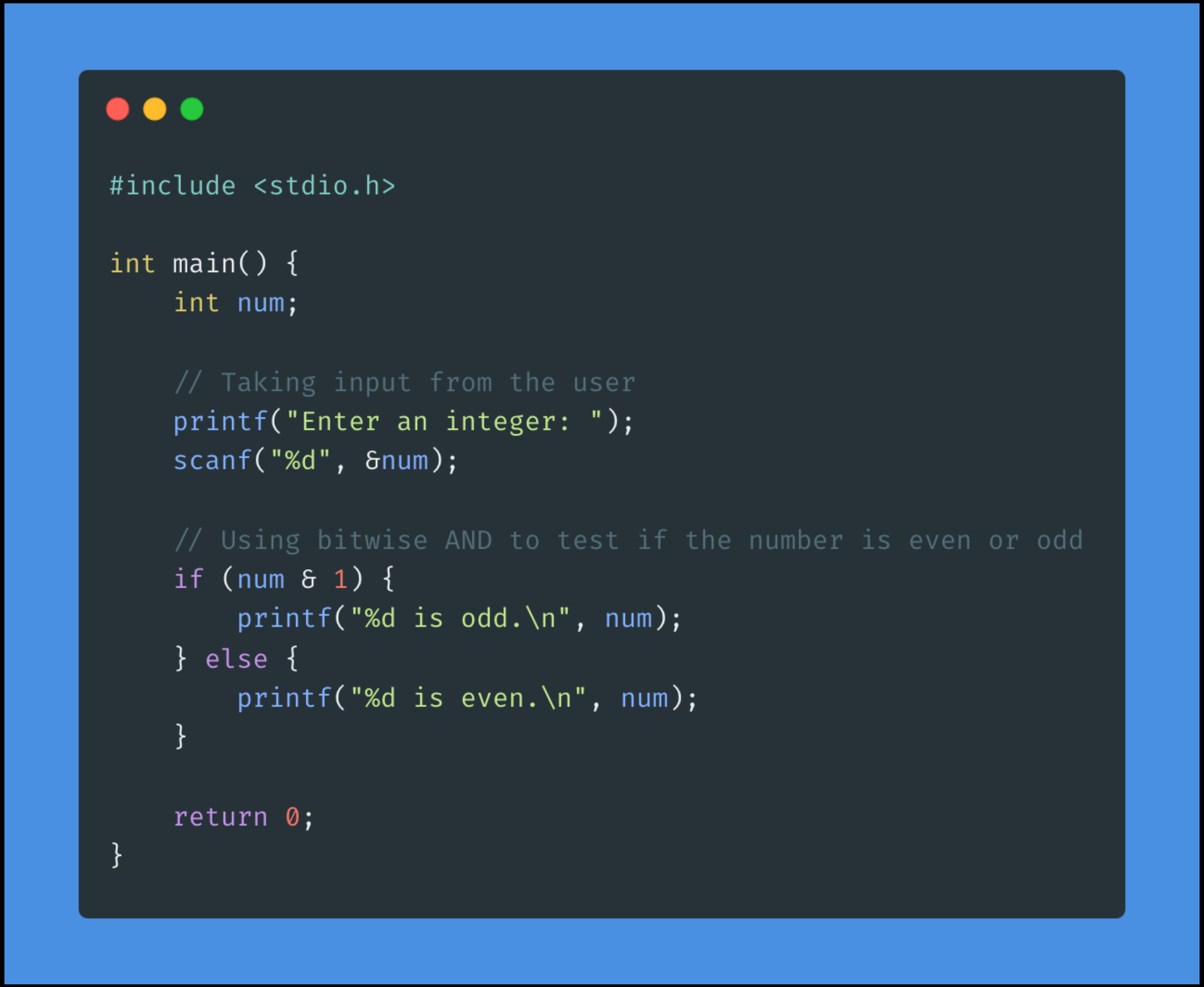
```c
#include <stdio.h>

// Function to check specific bits
int checkBits(int num, int mask) {
    return (num & mask) == mask;
}

int main() {
    int num = 29; // Binary: 00011101
    int mask = 4; // Binary: 00000100

    if (checkBits(num, mask)) {
        printf("Bits set by mask %d (binary: 00000100) are ON in number %d (binary: 00011101).\n", mask,
num);
    } else {
        printf("Bits set by mask %d (binary: 00000100) are OFF in number %d (binary: 00011101).\n", mask,
num);
    }

    return 0;
}
```

Find Odd or Even with &



Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Find Odd or Even with &



```c
#include <stdio.h>

int main() {
    int num;

    // Taking input from the user
    printf("Enter an integer: ");
    scanf("%d", &num);

    // Using bitwise AND to test if the number is even or odd
    if (num & 1) {
        printf("%d is odd.\n", num);
    } else {
        printf("%d is even.\n", num);
    }

    return 0;
}
```

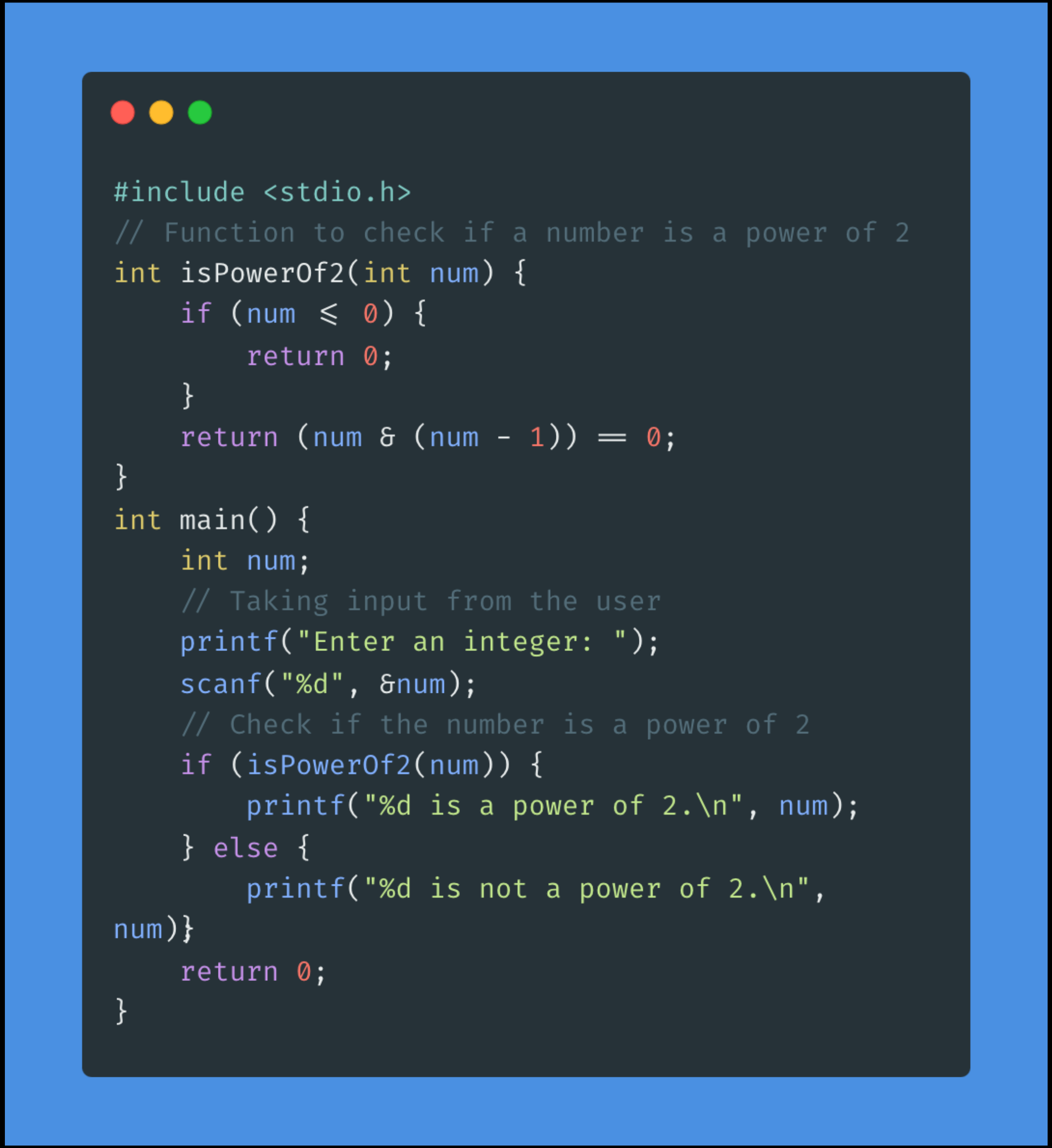
Check if power of 2

8 = 00001000
7 = 00000111

16 = 00010000
15 = 00001111

32 = 00100000
31 = 00011111

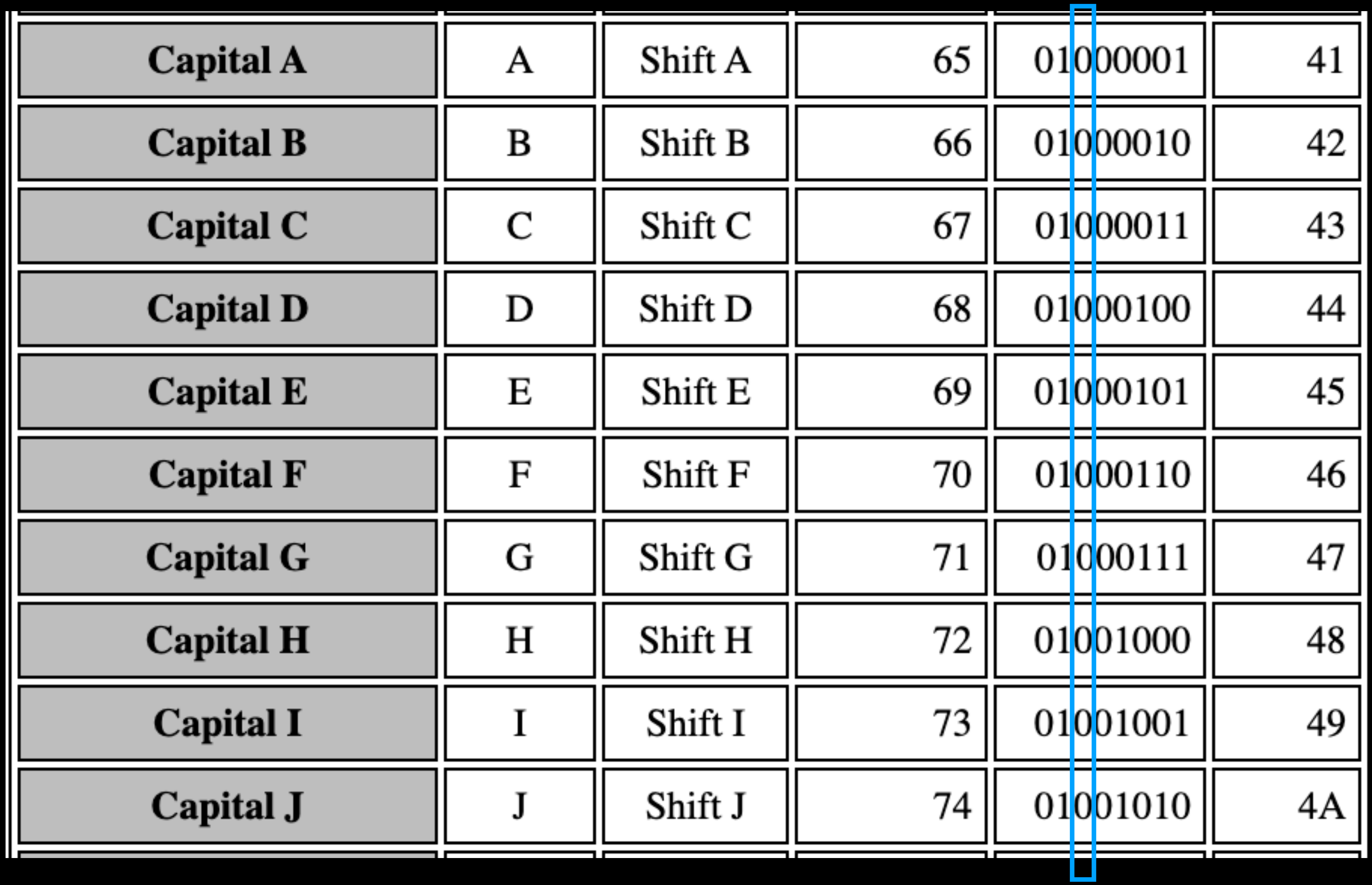
Check if power of 2



```c
#include <stdio.h>
// Function to check if a number is a power of 2
int isPowerOf2(int num) {
    if (num <= 0) {
        return 0;
    }
    return (num & (num - 1)) == 0;
}
int main() {
    int num;
    // Taking input from the user
    printf("Enter an integer: ");
    scanf("%d", &num);
    // Check if the number is a power of 2
    if (isPowerOf2(num)) {
        printf("%d is a power of 2.\n", num);
    } else {
        printf("%d is not a power of 2.\n",
num)}
    return 0;
}
```

TO LOWERCASE

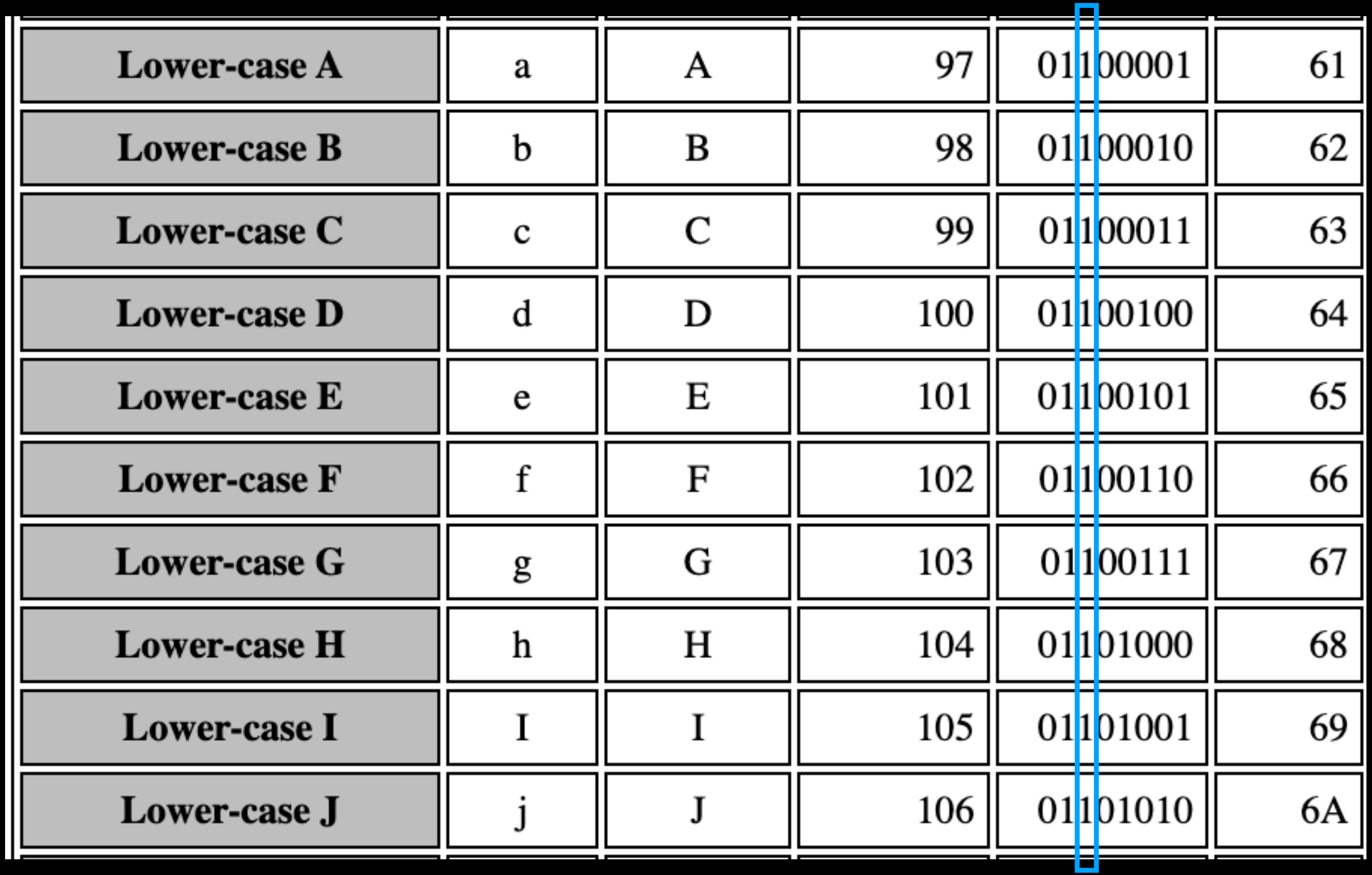
32 = 00100000



Capital A	A	Shift A	65	01000001	41
Capital B	B	Shift B	66	01000010	42
Capital C	C	Shift C	67	01000011	43
Capital D	D	Shift D	68	01000100	44
Capital E	E	Shift E	69	01000101	45
Capital F	F	Shift F	70	01000110	46
Capital G	G	Shift G	71	01000111	47
Capital H	H	Shift H	72	01001000	48
Capital I	I	Shift I	73	01001001	49
Capital J	J	Shift J	74	01001010	4A

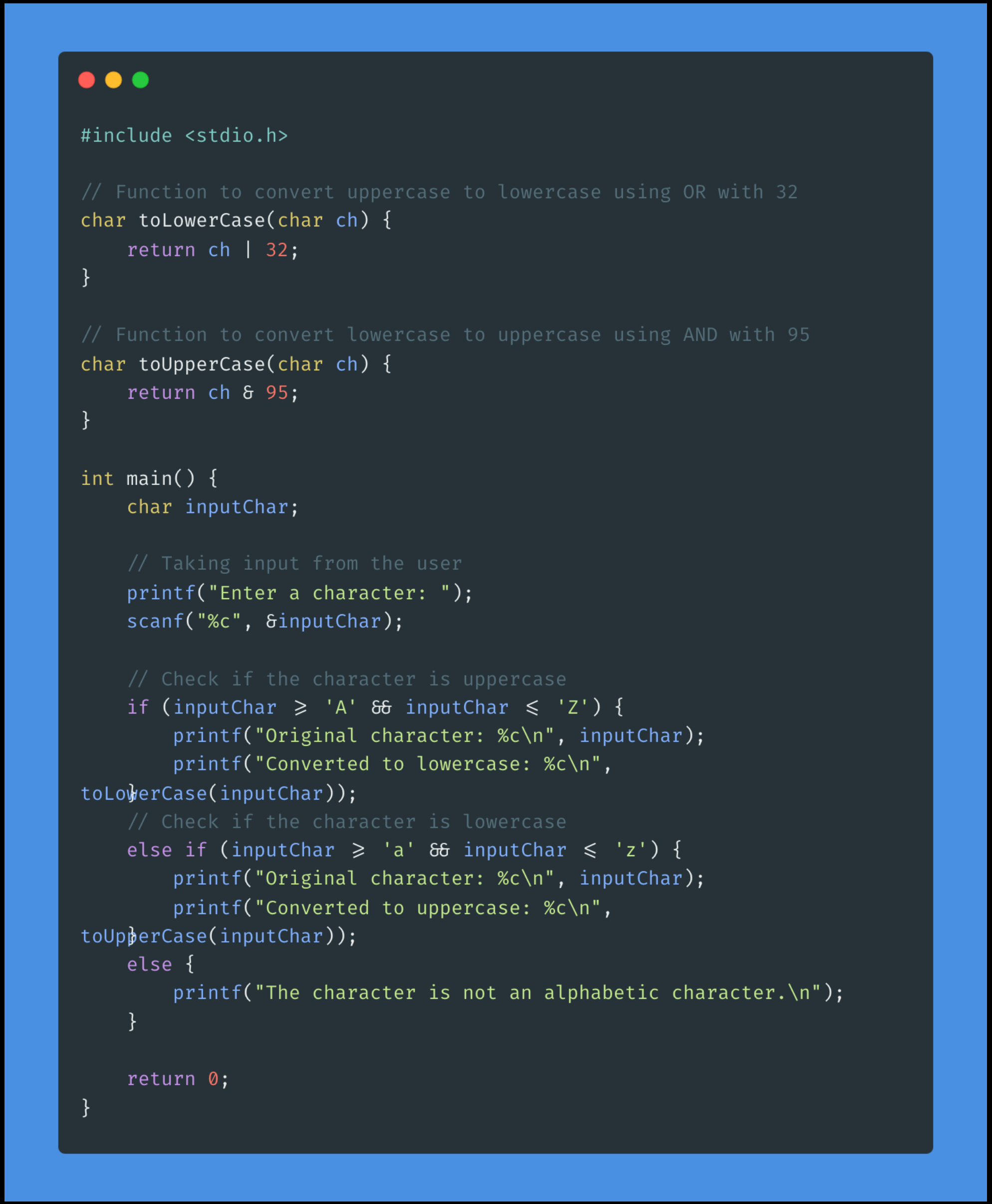
TO LOWERCASE

95 = 01011111



Lower-case A	a	A	97	01100001	61
Lower-case B	b	B	98	01100010	62
Lower-case C	c	C	99	01100011	63
Lower-case D	d	D	100	01100100	64
Lower-case E	e	E	101	01100101	65
Lower-case F	f	F	102	01100110	66
Lower-case G	g	G	103	01100111	67
Lower-case H	h	H	104	01101000	68
Lower-case I	I	I	105	01101001	69
Lower-case J	j	J	106	01101010	6A

Changing Case



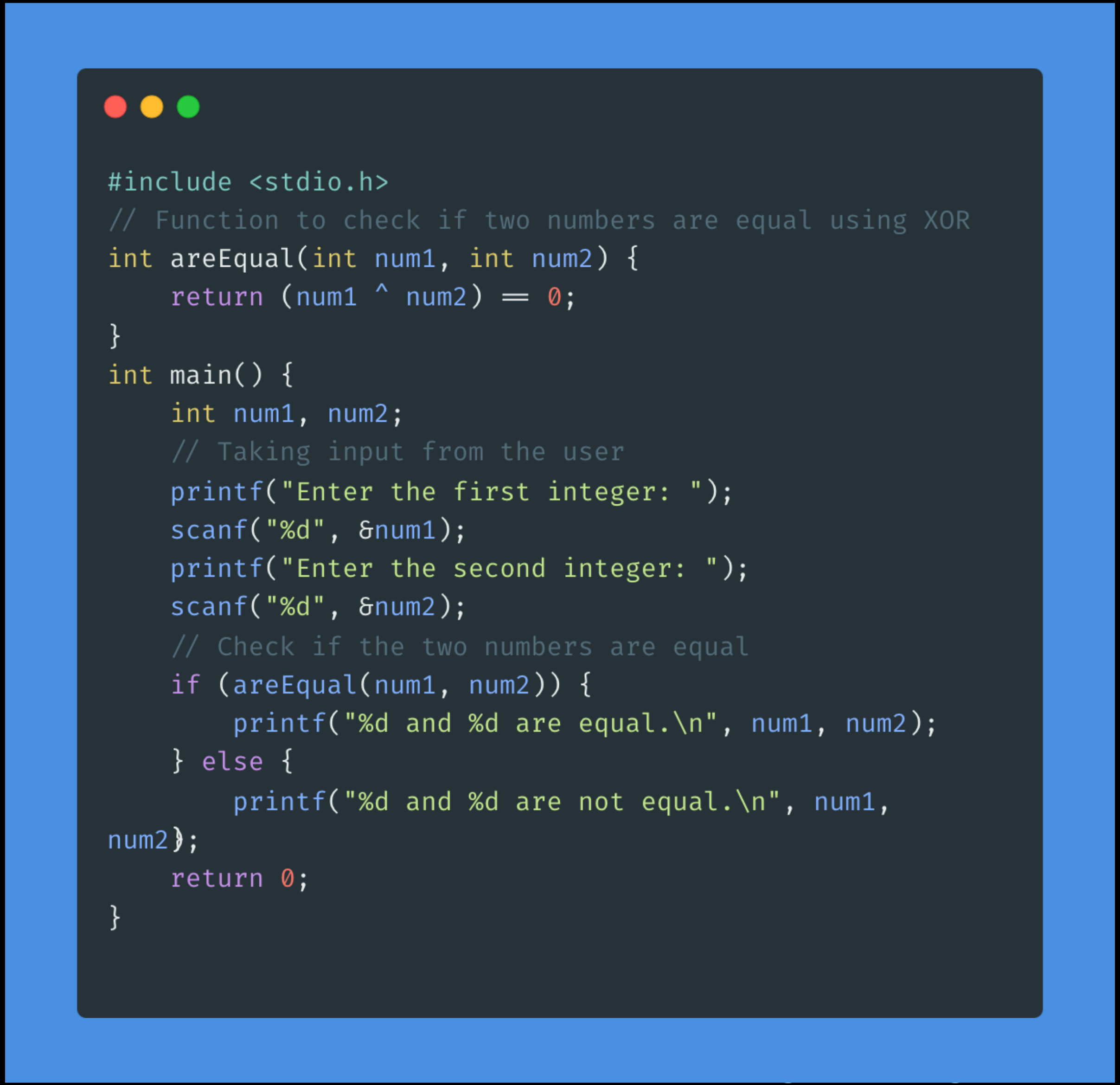
```c
#include <stdio.h>

// Function to convert uppercase to lowercase using OR with 32
char toLowerCase(char ch) {
    return ch | 32;
}

// Function to convert lowercase to uppercase using AND with 95
char toUpperCase(char ch) {
    return ch & 95;
}

int main() {
    char inputChar;

    // Taking input from the user
    printf("Enter a character: ");
    scanf("%c", &inputChar);

    // Check if the character is uppercase
    if (inputChar >= 'A' && inputChar <= 'Z') {
        printf("Original character: %c\n", inputChar);
        printf("Converted to lowercase: %c\n",
toLowerCase(inputChar));
    }
    // Check if the character is lowercase
    else if (inputChar >= 'a' && inputChar <= 'z') {
        printf("Original character: %c\n", inputChar);
        printf("Converted to uppercase: %c\n",
toUpperCase(inputChar));
    }
    else {
        printf("The character is not an alphabetic character.\n");
    }

    return 0;
}
```

Programming Course

Check if two numbers are equal

Property of XOR:

A xor A = 0
So if two numbers are same
There xor will be 0.



```c
#include <stdio.h>
// Function to check if two numbers are equal using XOR
int areEqual(int num1, int num2) {
    return (num1 ^ num2) == 0;
}
int main() {
    int num1, num2;
    // Taking input from the user
    printf("Enter the first integer: ");
    scanf("%d", &num1);
    printf("Enter the second integer: ");
    scanf("%d", &num2);
    // Check if the two numbers are equal
    if (areEqual(num1, num2)) {
        printf("%d and %d are equal.\n", num1, num2);
    } else {
        printf("%d and %d are not equal.\n", num1,
num2);
    return 0;
}
```

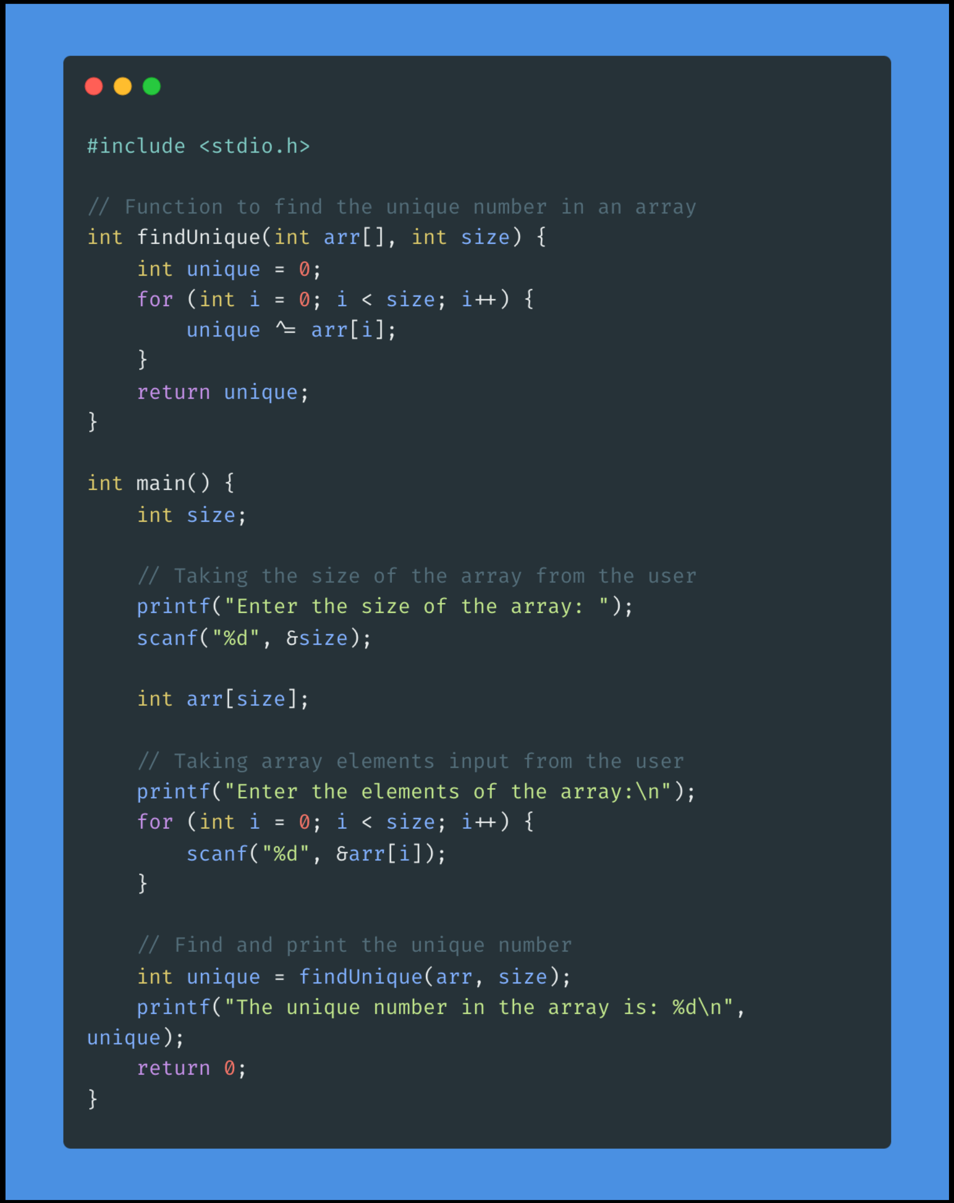
Find Unique in an array

Property of XOR:

A xor A = 0
So numbers appearing twice or
Multiple of two times will become
0.
B xor 0 = B
The remaining unique number will be
The result.

Note:

Every other number in the array must
Appear twice or even times to find the
Unique number.



```c
#include <stdio.h>

// Function to find the unique number in an array
int findUnique(int arr[], int size) {
    int unique = 0;
    for (int i = 0; i < size; i++) {
        unique ^= arr[i];
    }
    return unique;
}

int main() {
    int size;

    // Taking the size of the array from the user
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int arr[size];

    // Taking array elements input from the user
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    // Find and print the unique number
    int unique = findUnique(arr, size);
    printf("The unique number in the array is: %d\n",
unique);
    return 0;
}
```