Bitwise operators

Some Content + Practice codes as well

Content:

1) Bitwise operators (some more intro)
2) Practice Codes (from past papers and assignments)

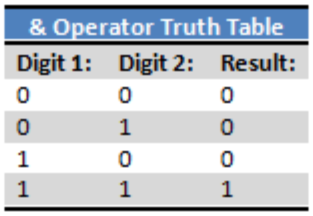
Bitwise operators:

Six bitwise operators:

Operator	Symbol	Form	Operation
left shift	<<	x << y	all bits in x shifted left y bits
right shift	>>	x >> y	all bits in x shifted right y bits
bitwise NOT	~	~x	all bits in x flipped
bitwise AND	&	x & y	each bit in x AND each bit in y
bitwise OR	|	x | y	each bit in x OR each bit in y
bitwise XOR	^	x ^ y	each bit in x XOR each bit in y

1. Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.



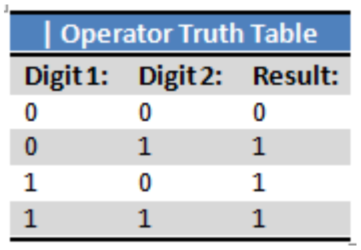
& Operator Truth Table		
Digit 1:	Digit 2:	Result:
0	0	0
0	1	0
1	0	0
1	1	1

Let us suppose the bitwise AND operation of two integers 12 and 25.

```
12 (In decimal) = 0000000000001100 (In Binary)


25 (In decimal) = 0000000000011001 (In Binary)




Bit Operation of 12 and 25


  0000000000001100


& 0000000000011001


  _________________


  0000000000001000 = 8 (In decimal)
```

2. Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C++ Programming, bitwise OR operator is denoted by |.

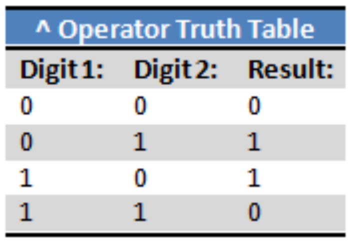


Operator Truth Table		
Digit 1:	Digit 2:	Result:
0	0	0
0	1	1
1	0	1
1	1	1

```
12 (In decimal) = 0000000000001100 (In Binary)

25 (In decimal) = 0000000000011001 (In Binary)

Bitwise OR Operation of 12 and 25

  0000000000001100

| 0000000000011001

  ________________

  0000000000011101  = 29 (In decimal)
```

3. Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite/different. It is denoted by ^.



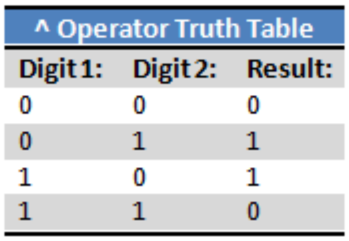
^ Operator Truth Table

Digit 1:	Digit 2:	Result:
0	0	0
0	1	1
1	0	1
1	1	0

```
12 = 0000000000001100 (In Binary)

25 = 0000000000011001 (In Binary)

Bitwise XOR Operation of 12 and 25

  0000000000001100

^ 0000000000011001

  ________________

  0000000000010101  = 21 (In decimal)
```

4. Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1, i.e., it gives the 1’s complement of a value. It is denoted by ~.



^ Operator Truth Table

Digit 1:	Digit 2:	Result:
0	0	0
0	1	1
1	0	1
1	1	0

```
35 (In decimal) = 0000000000100011 (In Binary)



Bitwise complement Operation of 35

~ 0000000000100011

  _________________

  1111111111011100  = -36 (In decimal)
```

//Execute the following codes and explain what is happened.

```
……………………….code1.cpp…………………………………..

int main()

{

        cout << (2.2 | 1.1);


}

……………………….code1a.cpp…………………………………..

int main()

{

        cout << (‘a’ | ‘b’);


}


……………………….code2.cpp…………………………………..

int main ()

{

        short  a=0;

        cout<<~a;

        return 0;

}

……………………….code3.cpp…………………………………..

int main ()

{
	short alpha=15;

	short beta = 245;

	cout<<endl;

	cout<< (alpha | beta)<<endl;

	cout<< (alpha & beta)<<endl;

	cout<< ~alpha<<endl;

	cout<< ~beta <<endl;

	cout<< (alpha ^ beta)<<endl;

	return 0;

}
```

……………………….code4.cpp…………………………………….

```
int main ()

{

	short a=1;

	a=a<<1;

	cout<<a;

	a=a<<1;

	cout<<a;

	return 0;

}
```

……………………….code5.cpp…………………………………….

```
int main ()

{

	short a=1;
```

```
            a=a<<15;

            cout<<a;

            return 0;

}
```

………………………..code5a.cpp…………………………………….

```
int main() {

            short a = 1;

            a = (a << 18);

            cout << a;

}
```

Tasks:

1) Compile all code segments (1 to 4). Write the appropriate reason of the output. (Write outputs on paper appropriately using binary values)

2) Rewrite all code segments (1 to 4) by using hexadecimal data now compare outputs with problem 1's outputs. (Write outputs on paper appropriately using binary values)

3) Rewrite all code segments (1 to 4) by using octal data now compare outputs with problem 1's and 2's outputs.

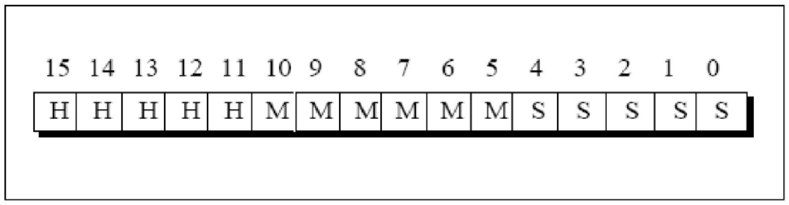
4) Value masking using Bitwise OR operator (Masking bits to 1/ **MASKED ON**):

To turn certain bits on, the bitwise OR operation can be used, following the principle that Y OR 1 = 1 and Y OR 0 = Y. Therefore, to make sure a bit is on, OR can be used with a 1. To leave a bit unchanged, OR is used with a 0.

Write a C++ program for OR masking using Bitwise OR operator.

Problems from previous Exams:

1) In order to save disk space Time field in the directory entry is 2 bytes long. Distribution of different bits, which account for hours, minutes and seconds, is given below:

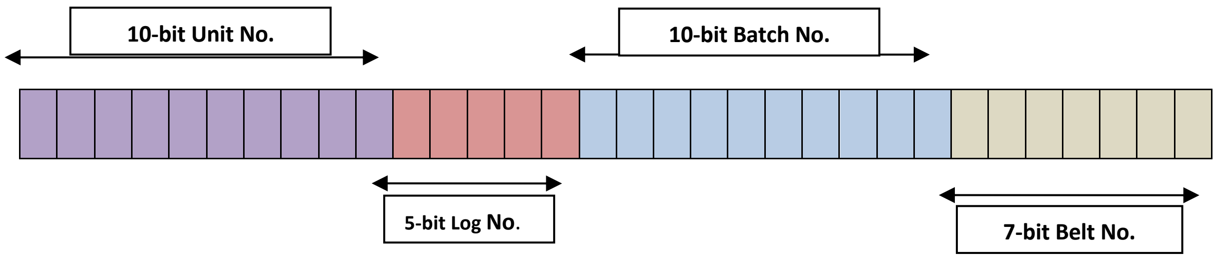


Write a C++ program that take input two-byte time entry and appropriately separates hours, minutes and seconds using **bitwise shift operators** only.

2) In a Military database system ID of an army man is stored in to a 32-bit value which is actually composed of four separate values these are:

- 7-bits Belt number
- 10-bits Batch number
- 5-bits Log number and
- 10-bits Unit number

And stored in following format:



Your Task is to write a C++ Program that take input four-byte ID and a string as Name of army Man. Your Program appropriately separates Belt number, Batch number, Log number and Unit number and prints output is following manner.

Sample Execution of Program:

```
Enter Name of Army Man:  Abdullah Khan
Enter Name of Army Man:  4444444

Belt number of Abdullah Khan is :      1
Batch number of Abdullah Khan is:      1
Log number  of Abdullah Khan is:    930
Unit number of Abdullah Khan is:     28
```

3) The information about colors is to be stored in bits of a variable called color. The bit number 0 to 6, each represent 7 colors of a rainbow, i.e. bit 0 represents violet, 1 represents indigo, and so on. Write a C++ program that asks the user to enter a number and based on this number it reports which colors in the rainbow does the number represents. If you need you can use single selection structure here.

Assignment Problem:

Scenario:

You are a junior security consultant at ***SecureTech*** *Solutions*, a company providing cybersecurity services to various clients. Recently, one of the company's major clients, ***SafeBank***, has come under scrutiny due to weak security protocols. Your boss has assigned you a series of tasks aimed at *strengthening* SafeBank's security systems. You must use *basic programming techniques*, such as *arithmetic and bitwise operations*, to improve encryption, data masking, payroll security, and data transmission.

Part #1: Message Encryption with ASCII Shift and Modulo Operation

SafeBank wants to secure customer communications by encrypting messages. Your boss has asked you to design a simple encryption algorithm using a combination of ASCII shifting and modulo operations to wrap around the character set.

Instructions:
Write a program that takes a single character as input, shifts its ASCII value 5 positions forward, and ensure the result stays within the ASCII range. Additionally, write code to decrypt the character by reversing the process.

Example Input/Output:

```
Enter a character to encrypt: A
Encrypted character: F
Decrypted character: A
```

Note: Do not use any conditional statement.

Part #2: Secure Salary Calculation

SafeBank's payroll system needs to perform more complex salary calculations. You need to calculate the final salary by applying a security multiplier of 1.5, then adding a security bonus of 500. After that, you must apply a tax deduction of 10% on the total salary.

Instructions:
Write a program that takes the basic salary as input, multiplies it by 1.5, adds 500 as a security bonus, and then deducts 10% from the total to calculate the final salary after tax.

Example Input/Output:

```
Enter the basic salary: 2000
Final salary after tax: 3150
```

Note: Calculate the final salary cleverly.

Part #3: Securing Data Transmission with rotation

When transmitting data between servers, SafeBank uses bitwise rotation to scramble the data. Your job is to implement both left and right rotation by 3 bits and then combine the two results to create a more complex scrambled value.

Instructions:
Write a program that takes an 8-bit integer as input, performs a left rotation by 3 bits, then a right rotation by 3 bits on the original value (keep in mind the overflow of bits), and finally applies a bitwise XOR between the two rotated values. Output the result.

Example Input/Output:

```
Enter an 8-bit value: 37
Result after combining rotations:
141
```

Note: Use bitwise operators to handle overflow as well.

Part #4: XOR-based Encryption

SafeBank uses XOR encryption for certain types of sensitive data. This time, instead of applying XOR just once, you must apply multiple transformations using different keys.

Instructions:
Write a program that takes a character as input, applies an XOR encryption with three different keys of user's choice. Reverse the process to decrypt the character.

Example Input/Output:

```
Enter a character to encrypt: A
Encrypted character: J
Decrypted character: A
```

Part #5: Security bit check

SafeBank uses a byte to store flags, where each bit represents a different security setting. You need to check if specific flags (2nd and 5th bits) are set, and then toggle the 6th bit. (Toggling is to invert the bit, if 0 then set it to 1 and if 1 then reset it to 0)

Instructions:
Write a program that takes an 8-bit integer as input, checks if the 2nd and 5th bits are set, and then toggles the 6th bit. Output the final byte.

Example Input/Output:

```
Enter an 8-bit value: 85
2nd bit is 1 (1 for set, 0 for not set).
5th bit is 0 (1 for set, 0 for not set).
Value, after toggling the 6th bit: 117
```

Note: Use bitwise AND, OR, and XOR to manipulate the bits.

Part #6: Parity Check

SafeBank wants to ensure data integrity through parity checks. In addition to checking if the number of 1-bits is odd or even, you must also perform an operation to invert all the bits in the input after the check.

Instructions:
Write a program that takes an 8-bit unsigned integer as input, determines whether the number of

1-bits is odd or even (parity), and then inverts all the bits. Output both the parity and the inverted value.

Example Input/Output:

```
Enter an 8-bit value: 37
Parity = 0.
Inverted value: 26
```

Here *Parity = 0* shows that it's an odd parity.

Part #7: Multiplication Using Shifts

SafeBank wants to avoid floating-point operations in its system, so multiplication by powers of 2 is achieved. Additionally, after multiplying a number, you need to add a fixed value of 250 to ensure the result falls within a secure range.

Instructions:
Write a program that takes an integer as input, multiplies it by 16, and then adds 250 to the result. Output the final value.

Example Input/Output:

```
Enter a number: 7
Final Result : 362
```

Note: For multiplication, '*' is **not** allowed.

Part #8: Modular Arithmetic

SafeBank uses modular arithmetic for secure transaction processing.

Instructions:
Write a program that takes an integer as input, calculates the remainder when dividing by 256, adds 100 to the remainder, and then applies another modulo operation with 256. Output the final result.

Example Input/Output:

```
Enter a number: 1000
Final result: 76
```