

OOP

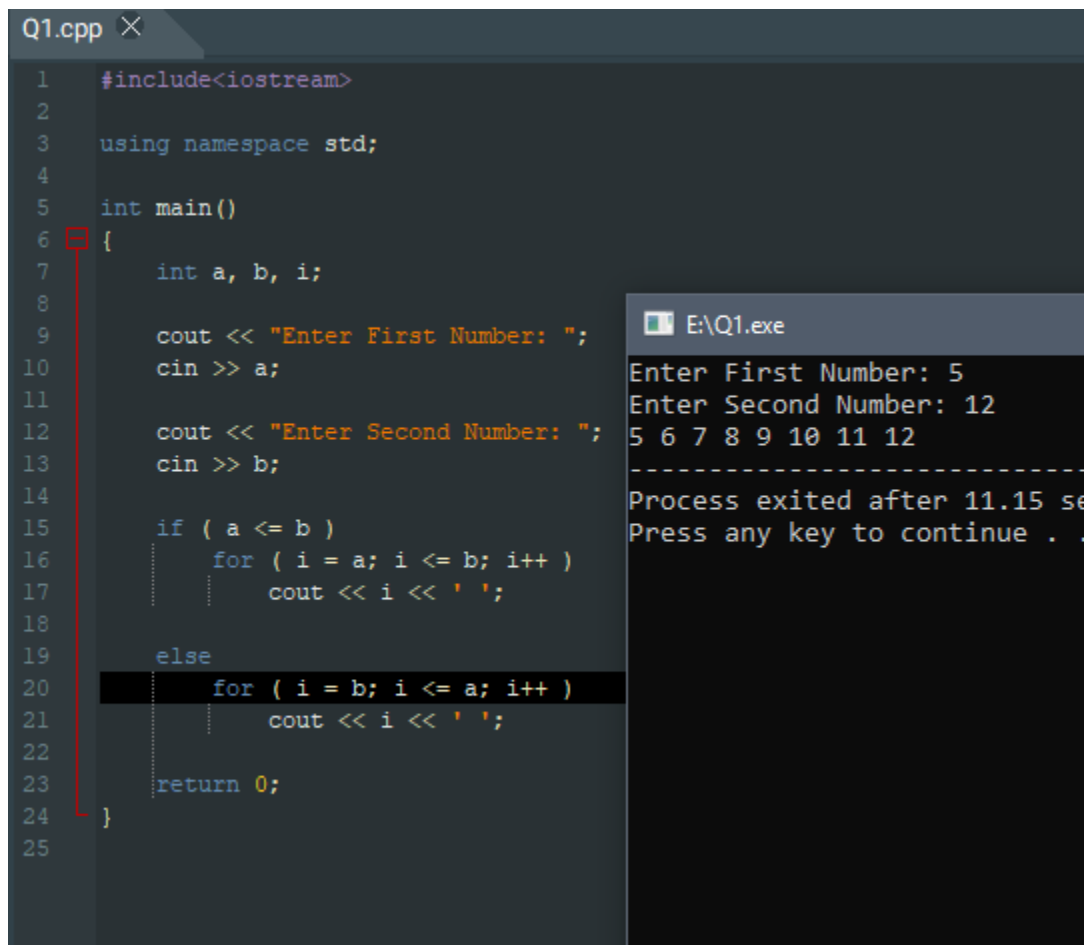
Lecture 5

21/10/2021

Quiz 1 was held in lecture 5.

Quiz 1:

1. Take 2 numbers from the user and print all natural numbers between them in ascending order.



The screenshot shows a C++ IDE with a file named Q1.cpp. The code is as follows:

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int a, b, i;
8
9      cout << "Enter First Number: ";
10     cin >> a;
11
12     cout << "Enter Second Number: ";
13     cin >> b;
14
15     if ( a <= b )
16     {
17         for ( i = a; i <= b; i++ )
18             cout << i << ' ';
19     }
20     else
21     {
22         for ( i = b; i <= a; i++ )
23             cout << i << ' ';
24     }
25     return 0;
26 }
```

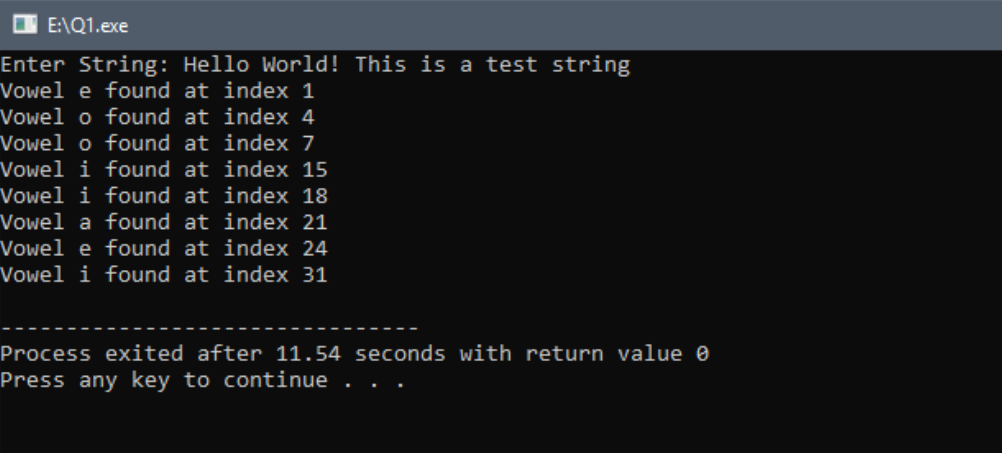
Next to the code editor is a terminal window titled E:\Q1.exe. It shows the program's execution:

```
Enter First Number: 5
Enter Second Number: 12
5 6 7 8 9 10 11 12
-----
Process exited after 11.15 seconds
Press any key to continue . .
```

Here, we need to check first which number is greater and then we can print all the numbers in ascending order.

2. Print all the vowel letters and their indexes from a string.

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      char str[100];
8      int i;
9
10     cout << "Enter String: ";
11     cin.getline(str, 100, '\n');
12
13     for ( i = 0; str[i] != '\0'; i++ )
14     {
15         if ( str[i] >= 65 && str[i] <= 90 )
16             str[i] += 32;
17
18         if ( str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' )
19             cout << "Vowel " << str[i] << " found at index " << i << endl;
20     }
21
22     return 0;
23 }
24
```



Here, the problem arises with the Capital and small letters as we need to consider both. So, we have two options: either we can add checks for the capital and small letters both or we can convert all the letters in one format.

1. For first case we need to write such statement:

```
if ( str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' || str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U' )
```

But, we can see that it became a long statement to read and the number of checks doubled. So, it is better to go with the second option.

Now, we need to convert all the letters in one format.

As we know the ASCII value of 'A' is 65 and 'a' is 97. There is a difference of 32 in both letters. Similarly ASCII of 'B' is 66 and 'b' is 98, 'C' is 67 and 'c' is 99 and so on. The difference of 32 just continues. It implies that if we just add 32 in capital letters it will become small, and similarly if we subtract 32 from capital letters it will become small letters.

3. Find the Index and Length of the largest string.

```
#include<iostream>

using namespace std;

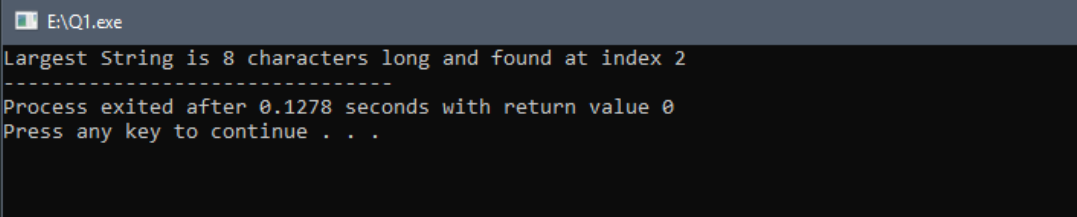
int main()
{
    char str[8][10] = { {"This"}, {"is"}, {"supposed"}, {"to"}, {"be"}, {"a"}, {"mock"}, {"string"} };
    int i, j, count, maxCount = 0, maxCountIndex;

    for ( i = 0; i < 8; i++ )
    {
        count = 0;
        for ( j = 0; str[i][j] != '\0'; j++ )
            count++;

        if ( maxCount < count )
        {
            maxCount = count;
            maxCountIndex = i;
        }
    }

    cout << "Largest String is " << maxCount << " characters long and found at index " << maxCountIndex;

    return 0;
}
```



E:\Q1.exe
Largest String is 8 characters long and found at index 2

Process exited after 0.1278 seconds with return value 0
Press any key to continue . . .

That was all about Quiz 1. After the discussion of the quiz. There was an introductory lecture on the bitwise operations. We will discuss it further details in the next lecture.

Extra Lecture 1

22/10/2021

Bitwise Operations:

Bitwise operators are used to perform bitwise operations on data. One byte is the lowest level at which we can access data. There is no "bit" type, and we can't access a single bit. In fact, we can't even perform operations on a single bit. Every bitwise operator will be applied to, at a minimum, an entire byte at a time. This means we'll be considering the whole representation of a number whenever we talk about applying a bitwise operator. (Note that this doesn't mean we can't ever change only one bit at a time, it just means we have to be smart about how we do it).

In the whole bitwise lecture we need to work on the binary representations of the numbers. Here we are using char data type because 1 character is of 1 byte(8 bits) so it will be easier to understand and manipulate the data. However, the same operations can be performed on any type of data.

There are a total of 6 Bitwise operators out of which 4 are logical operators and 2 are shift Operators.

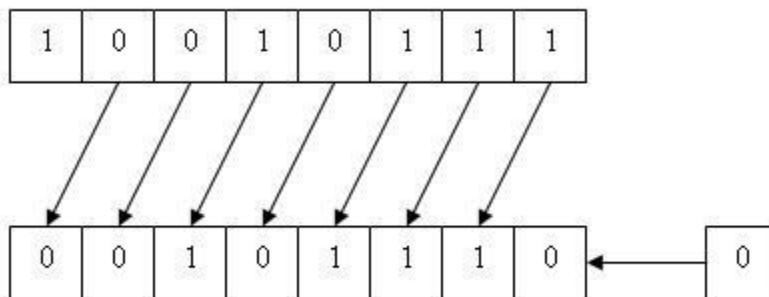
Shift Operators:

The shift operators are used to shift all the bits of a number towards left or right. There are 2 shift operators i.e

1. Left Shift Operator
2. Right Shift Operator

Left Shift Operator:

The left shift operator is used to shift all the bits of a number towards the left. For example, Consider a binary number: 01101100. If we apply left shift on this number the leftmost digit of this number will be excluded and all the bits will be shifted towards left and the result will be like 1101100_ with an empty space at the rightmost bit. Now the empty space will be filled with 0. And the resultant will be 11011000.



Grasp the idea of how this left shifting of bits works.

Syntax:

Operand << number_of_bits_to_shift;

Example:

```
8 << 1;  
20 << 3;  
x << 2;
```

Mathematical Behaviour:

If we check the mathematical analysis of the left shift operator we can clearly see that if a number is left shifted by one bit the resultant will be its double.

Left shift 1 means multiplication by 2, left shift 2 means multiplication by 4, Left shift 3 means multiplication by 8 and so on.

Example:

0 0 0 1 1 1 1 0 : 30 (in decimal)

Now, shifting 1 bit towards the left.

0 0 1 1 1 1 0 0 : 60 (in decimal)

```
#include <iostream>

using namespace std;

int main()
{
    //Left shift 1 means multiplication by 2, left shift 2 means multiplication by 4

    cout << "16 left shift 1 time: " << (16 << 1) << '\n';
    cout << "48 left shift 1 time: " << (48 << 1) << '\n';
    cout << "15 left shift 2 time: " << (15 << 2) << '\n';
    cout << "31 left shift 1 time: " << (31 << 1) << '\n';

    return 0;
}
```

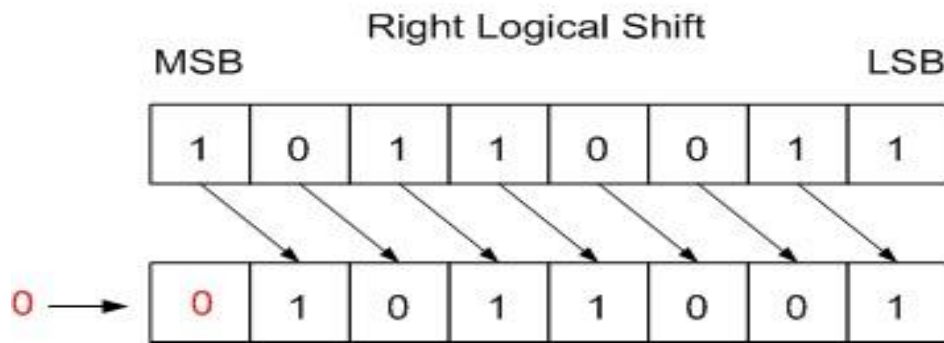
E:\Q1.exe

```
16 left shift 1 time: 32
48 left shift 1 time: 96
15 left shift 2 time: 60
31 left shift 1 time: 62
```

```
-----
Process exited after 0.07233 seconds with return value 0
Press any key to continue . . .
```

Right Shift Operator:

The right shift operator is used to shift all the bits of a number towards the right. For example, Consider a binary number: 01101100. If we apply a right shift on this number the rightmost digit of this number will be excluded and all the bits will be shifted towards right and the result will be like _0110110 with an empty space at the leftmost bit. Now the empty space will be filled with 0. And the resultant will be 00110110.



Grasp the idea of how this left shifting of bits works.

Syntax:

Operand >> number_of_bits_to_shift;

Example:

```
8 >> 1;
20 >> 3;
x >> 2;
```

Mathematical Behaviour:

If we check the mathematical analysis of the right shift operator we can clearly see that if a number is right shifted by one bit the resultant will be its half.

Right shift 1 means dividing by 2, right shift 2 means dividing by 4, Right shift 3 means dividing by 8 and so on.

Example:

0 0 0 1 1 1 1 0 : 30 (in decimal)

Now, shifting 1 bit towards the left.

0 0 0 0 1 1 1 1 : 15 (in decimal)

```
#include <iostream>

using namespace std;

int main()
{
    //Right shift 1 means division by 2, right shift 2 means division by 4

    cout << "16 right shift 1 time: " << (16 >> 1) << '\n';
    cout << "48 right shift 1 time: " << (48 >> 1) << '\n';
    cout << "48 right shift 2 time: " << (48 >> 2) << '\n';
    cout << "99 right shift 1 time: " << (99 >> 1) << '\n';

    return 0;
}
```

E:\Q1.exe

```
16 right shift 1 time: 8
48 right shift 1 time: 24
48 right shift 2 time: 12
99 right shift 1 time: 49

-----
Process exited after 0.0548 seconds with return value 0
Press any key to continue . . .
```

Bitwise Logical Operators:

There are 4 bitwise logical operators:

1. AND
2. OR
3. XOR
4. Complement

AND (&) :

Bitwise AND operator is a binary operator which means that it takes two operands and performs bitwise AND operations on them. The bitwise AND operation results 1 if and only if both the operands are 1. Otherwise, it results 0.

Truth table for AND operation is:

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Now, coming towards the bitwise AND. Bitwise means that it will perform AND operation on each corresponding bit of the operands. Consider 2 Numbers, 01110111 AND 10111010. If we perform bitwise AND on them

0 1 1 1 0 1 1 1		1 1 9	
& 1 0 1 1 1 0 1 0		& 1 8 6	(In Decimal)
-----		-----	
0 0 1 1 0 0 1 0		5 0	

Using bitwise AND operator we can turn off (reset) any specific bit of a number. We have to place 0 against the bit we want to reset and place 1 against the bits which we want to remain the same.

For Example, Consider 10111110. We want to turn off its 3rd and 6th bit then we should find a number like 11011011. Such that

```

    1 0 1 1 1 1 1 0
& 1 1 0 1 1 0 1 1
-----
    1 0 0 1 1 0 1 0 (3rd and 6th bit is off, rest is same)

```

Such a number is called Mask.

Mask:

A mask defines which bits you want to keep, and which bits you want to clear. Masking is the act of applying a mask to a value.

```

#include <iostream>

using namespace std;

// & operator is used to off the bits
int main()
{
    unsigned char c1 = 256 - 16, c2=255-32-8-1;

    //Now perform bit wise and
    unsigned char result = c1 & c2;

    cout << "c1: " << (int)c1 << " c2: " << (int)c2 << " Result: " << (int)result << "\n";

    return 0;
}

```

E:\Q1.exe

```

c1: 240 c2: 214 Result: 208

-----
Process exited after 0.1062 seconds with return value 0
Press any key to continue . . .

```

Here, 255 means 11111111. 255 - 16 means the bit with weight 16 (5th bit) is off and all other bits are off i.e 11101111.

255-32-8-1 means that 1st, 4th and 6th bits are off. i.e 11010110.

OR (|):

Bitwise OR operator is a binary operator which means that it takes two operands and performs bitwise OR operations on them.

The bitwise OR operation results in 1 if any of the operands is 1. And 0 if all operands are 0.

Truth table for OR operation is:

OR		
x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

Now, coming towards the bitwise OR. Bitwise means that it will perform OR operation on each corresponding bit of the operands. Consider 2 Numbers, 01110111 AND 10111010. If we perform bitwise AND on them

0 1 1 1 0 1 1 1		1 1 9	
1 0 1 1 1 0 1 0		1 8 6	(In Decimal)
-----		-----	
1 1 1 1 1 1 1 1		2 5 5	

Using bitwise OR operators we can turn ON (set) any specific bit of a number. We have to place 1 against the bit we want to set and place 0 against the bits which we want to remain the same.

For Example, Consider 01000010. We want to turn ON its 3rd and 6th bit then we should find a number like 00100100. Such that

```

0 1 0 0 0 0 1 0
& 0 0 1 0 0 1 0 0
-----
0 1 1 0 0 1 1 0 (3rd and 6th bit is turned ON, rest is same)

```

```

#include <iostream>

using namespace std;

// & operator is used to off the bits
int main()
{
    unsigned char c1 = 16, c2 = 8;

    //Now perform bit wise and

    unsigned char result = c1 | c2;

    cout << "c1: " << (int)c1 << " c2: " << (int)c2 << " Result: " << (int)result << "\n";

    return 0;
}

```

E:\Q1.exe

c1: 16 c2: 8 Result: 24

 Process exited after 0.1354 seconds with return value 0
 Press any key to continue . . .

Here are some Sample Programs discussed in class.

```
#include <iostream>

using namespace std;

int main()
{
    char c1 = 223, c2=8;
    //We want to check, whether bit 5 is on or off
    //We take a variable with some value to check 5th bit, technically it is called mask
    char mask = 16; //Binary of 16 is 00010000, where 5th bit is on and all other bits are off
    //Now perform bit wise and
    char result = c1 & mask; // & is bitwise and
    if (result == 0)    cout << "5th bit is off\n";
    else                cout << "5th bit is on\n";
    result = c2 & mask; // & is bitwise and
    if (result == 0)    cout << "5th bit is off\n";
    else                cout << "5th bit is on\n";
    return 0;
}
```

E:\Q1.exe

5th bit is on
5th bit is off

Process exited after 0.1552 seconds with return value 0
Press any key to continue . . .

```

#include <iostream>

using namespace std;

int main()
{
    char c1 = 223, c2=8;
    int bitNo;
    cout << "Enter Bit no (1-8):";
    cin >> bitNo;
    char mask = 1 << (bitNo-1) ;
    //Now perform bit wise and
    char result = c1 & mask; // & is bitwise and
    if (result == 0)    cout << "Bit no " << bitNo << " is off\n";
    else                cout << "Bit no " << bitNo << " is on\n";
    result = c2 & mask; // & is bitwise and
    if (result == 0)    cout << "Bit no " << bitNo << " is off\n";
    else                cout << "Bit no " << bitNo << " is on\n";
    return 0;
}

```

E:\Q1.exe

```

Enter Bit no (1-8):3
Bit no 3 is on
Bit no 3 is off

```

```

-----
Process exited after 3.932 seconds with return value 0
Press any key to continue . . .

```

```

#include <iostream>

using namespace std;

// | Bit-wise or is used to on the bits

void checkBits(unsigned char c){
    int bitNo = 1;
    unsigned char mask = 1, result;
    do{
        result = c & mask;
        if (result == 0)    cout << "bit no " << bitNo << " is off\n";
        else                cout << "bit no " << bitNo << " is on\n";
        if (mask==128) break;//Terminate the loop because after 128, number will be 0
        bitNo++;
        mask = mask << 1;
    }while (1);
}

int main()
{
    checkBits(223);
    cout << "-----\n";
    checkBits(210);
    return 0;
}

```

E:\Q1.exe

```

bit no 1 is on
bit no 2 is on
bit no 3 is on
bit no 4 is on
bit no 5 is on
bit no 6 is off
bit no 7 is on
bit no 8 is on

```

```

-----
bit no 1 is off
bit no 2 is on
bit no 3 is off
bit no 4 is off
bit no 5 is on
bit no 6 is off
bit no 7 is on
bit no 8 is on

```

```

-----
Process exited after 0.1798 seconds with return code 0
Press any key to continue . . .

```