CMP 1001

Introduction to programming

Part 9: Bitwise operators & File operations

by

Assist. Prof. M. Şükrü Kuran

BITWISE OPERATORS TRUTH TABLE

Bitwise AND

Bit a	Bit b	a AND b (a & b)
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR

Bit a	Bit b	a OR b (a | b)
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise XOR

Bit a	Bit b	a XOR b (a ^ b)
0	0	0
0	1	1
1	0	1
1	1	0

NOTE: "&&"is Logical AND operator. '&' is bitwise AND operator.
"||"is Logical OR operator. '|' is bitwise OR operator.
Mixing these up inside if and while statements is a COMMON MISTAKE

BITWISE OPERATORS

Bitwise AND (&) Operator

```
int numA = 11;
```

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

```
int numB = 23;
```

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

```
int numC = numA & numB;
```

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1

BITWISE OPERATORS

Bitwise OR (|) Operator

```
int numA = 11;
```

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

```
int numB = 23;
```

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

```
int numC = numA | numB;
```

7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	1

BITWISE OPERATORS

Bitwise XOR (^) Operator

```
int numA = 11;
```

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

```
int numB = 23;
```

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	1

```
int numC = numA ^ numB;
```

7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0

BITWISE OPERATORS

Shift Left (<<) Operator

```
int numA = 11;
```

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

```
int numC = numA << 1;
```

7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	0

```
int numD = numA << 2;
```

7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	0

BITWISE OPERATORS

Shift Right (>>) Operator

```
int numA = 11;
```

7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1

```
int numC = numA >> 1;
```

7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1

```
int numD = numA >> 2;
```

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0

BITWISE OPERATORS

Bitwise NOT (~) Operator

```
short numA = 11;
```

31	…	5	4	3	2	1	0
0	…	0	0	1	0	1	1

```
short numC = ~numA;
```

31	…	5	4	3	2	1	0
1	…	1	1	0	1	0	0

BITWISE OPERATORS

Example

Example:
Write a program that gets a sequence of integer numbers until "0" has been given. Check if each one of these integer numbers are odd or even numbers using the & operator.

Example:
Write a program that gets a sequence of integer numbers until "0" has been given. Multiply every even numbered number in the sequence by 4 and divide every odd numbered number in the sequence by 4 using the >> and << operators.

BITWISE OPERATORS

Example

Example:

Write a program that first reads an integer number (num), and another integer number representing the number of digits of a mask value (maskDigitCount). Next, the program will build up a mask based on the maskDigitCount as below:

maskDigitCount = 4, mask = $2^4 - 1 = 15$
maskDigitCount = 6, mask = $2^6 - 1 = 63$
maskDigitCount = 7, mask = $2^7 - 1 = 127$

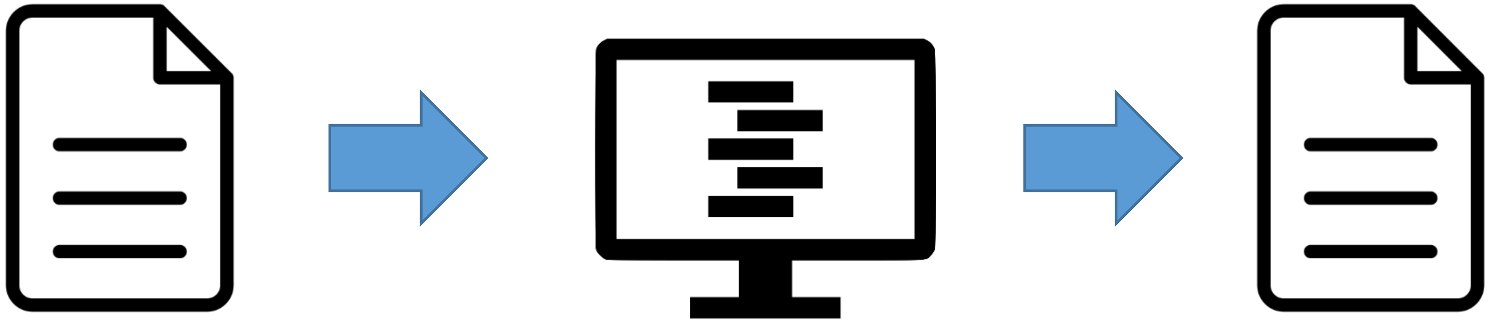
Finally, the program will apply the mask over the num via the bitwise AND operator to calculate the masked value of the num, and print it out to the console.

FILE OPERATIONS

Using Files as Input and Output sources

INPUT: Keyboard

OUTPUT: Monitor/Console



INPUT: File

OUTPUT: File

OFSTREAM

Output File Stream

We use a new variable "**std::ofstream**" to write something to a file.

```
#include <fstream>
...
std::ofstream outputFile;
```

Next, we have to open the file while providing the name of the file

```
outputFile.open("output.txt");
```

$\longrightarrow$ Relative Address

```
outputFile.open("D:\\output.txt");
```

$\longrightarrow$ Absolute Address

OFSTREAM

Output File Stream

We write inside this file similar to using std::cout.

```
int numA = 6;
outputFile << “Greetings” << numA;
```

Finally, we close the file when we finish writing to the file.

```
outputFile.close();
```

IFSTREAM

Input File Stream

We use a new variable “**std::ifstream**” to read something from a file.

```
#include <fstream>
...
std::ifstream inputFile;
```

Next, we have to open the file while providing the name of the file

```
inputFile.open(“output.txt”);
```

$\longrightarrow$ Relative Address

```
inputFile.open(“D:\\output.txt”);
```

$\longrightarrow$ Absolute Address

IFSTREAM

Input File Stream

We read from this file similar to using std::cin by reading to a string variable.

```
std::string s;
inputFile >> s;
```

```
getline(inputFile, s);
```

Finally, we close the file when we finish writing to the file.

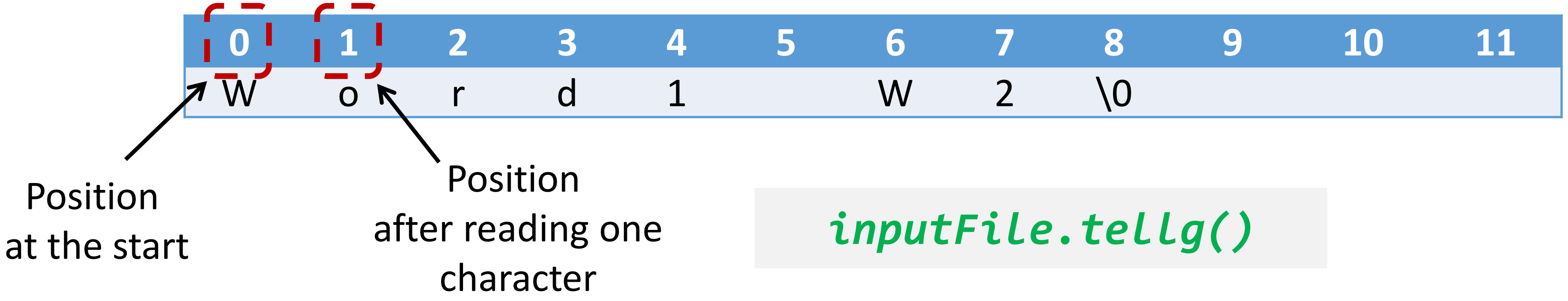
```
inputFile.close();
```

POSITION WITHIN FILE

Position Within an std::ifstream file

How do we keep track of where we are within the file while reading from it?

```
outputFile << “Word1 ” << “W2”;
```



0	1	2	3	4	5	6	7	8	9	10	11
W	o	r	d	1		W	2	\0			

```
inputFile.tellg()
```

Can we become aware if we have reached the end of a file?

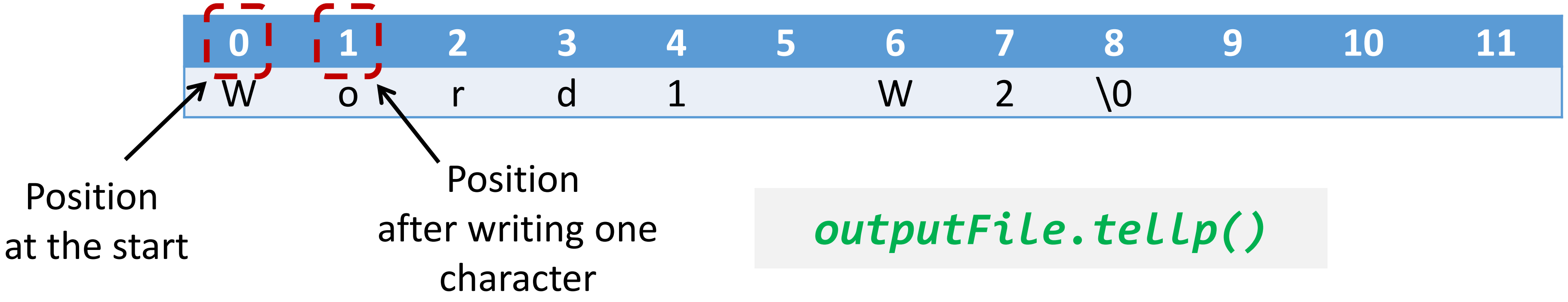
```
while (!inputFile.eof())
```

POSITION WITHIN FILE

Position Within an std::ofstream file

How do we keep track of where we are within the file while writing to it?

```
outputFile << “Word1 ” << “W2”;
```



```
outputFile.tellp()
```

FILE OPERATIONS

Example

Example:

Write a program that gets a sequence of characters until the ‘ *’ character is met. Then, the program will record the NEXT character within the ASCII table of ALL these characters (excluding the last ‘*’ character) to a file named “sequence.txt”. Lastly, the program will read back this “sequence.txt” file and print out all of the read characters to the console.

FILE OPERATIONS

Example

Example:

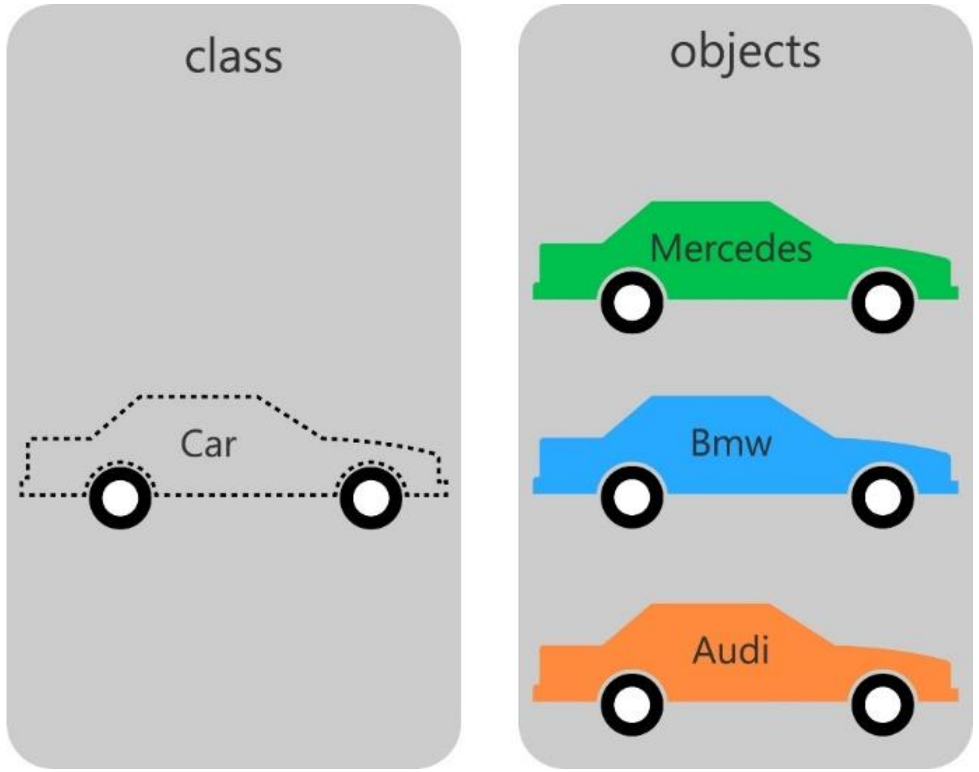
Open up a file named “numberSequence.txt” and fill it up with various integer numbers, delimited by a blank character such as below:

6 7 12 95 -7 -8 0 9 96 55 0 -6 -75
65 4 5 1 2 2 8

Write a program that reads this “numberSequence.txt” file and print out all of the EVEN read characters to the console.

COMING SOON...

Next week on CMP 1001



Classes & objects