Logical Operators, Operator Precedence, continue statement

Break

```
#include <iostream.h>
int main() {
 for ( int count = 1; count <= 10; count++ ) {
        if ( count == 5 ) // if count is 5,
                 break; // terminate from loop
        cout << count << " ";
   } // end for
cout << "\ nBroke out of loop at count = " <<
   count<< endl;
return 0; // indicate successful termination
} // end main
```

Continue

```
#include <iostream.h>
int main() {
 for ( int count = 1; count <= 10; count++ ) {
        if ( count == 5 ) // if count is 5,
                continue; // skip remaining code in loop
        cout << count << " ";
   } // end for
cout << "\nUsed continue to skip printing 5" <<
   endl;
return 0; // indicate successful termination
} // end main
```

The continue statement

- If we want to come out of the loop we use break statement, similarly if we want to go back to the top of the loop, we use continue statement.

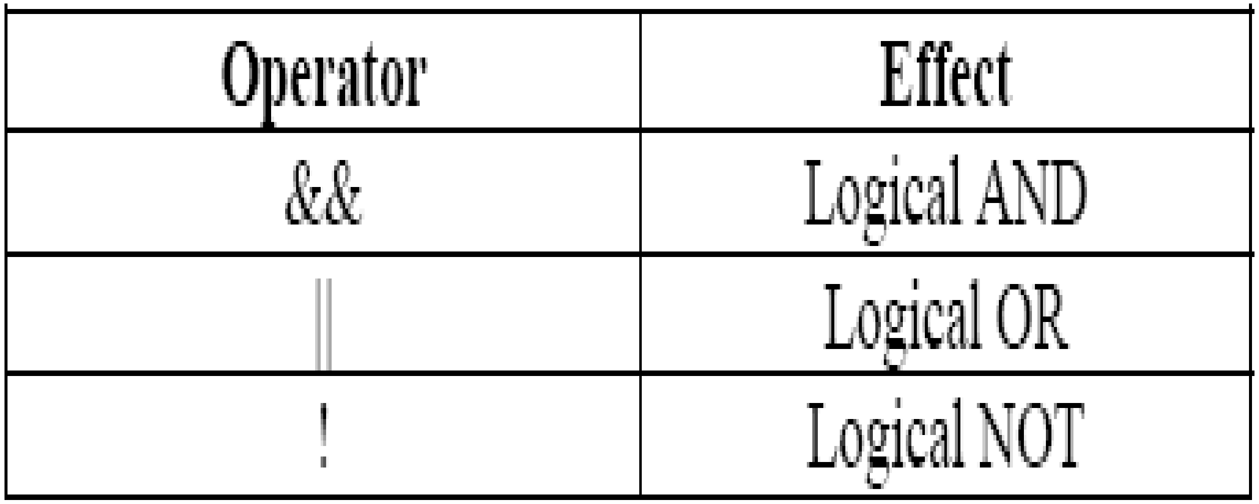
The continue statement

```
#include <iostream.h> #include <conio.h>
void main(void)
{ long divident, divisor;
char ch;
do {
     clrscr();
     cout<<"Enter Divident ";cin>>divident;
     cout<<"Enter Divisor ";cin>>divisor;
     if(divident==0 || divisor==0)
        {
                 cout<<"Divident or Divisor can not be 0";
                 getch();
                 continue;
          }
     cout<<divident<<" / "<<divisor<<" = "<<divident / divisor<<endl;
     cout<<divident<<" % "<<divisor<<" = "<<divident % divisor<<endl;
     cout<<"Continue ";
     ch=getche();
     }while(ch!='n');

cout<<"\nProgram Finished";
getch();
}
```

Logical Operators

- Logical operators allow the programmer to combine Boolean (True/False) values.



Operator	Effect
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical Operators

```
#include<iostream.h>
#include<conio.h>
void main(void)
{               clrscr();
                int p;
                do
                {               cout<<"Enter your percentage (0-100) ";
                                cin>>p;
                }
                while(p<0 || p>100);
                if(p>=90)
                                cout<<p<<"% = A+ Grade";
                else if(p>=80 && p<90)
                                cout<<p<<"% = A  Grade";
                else if(p>=70 && p<80)
                                cout<<p<<"% = B  Grade";
                else if(p>=60 && p<70)
                                cout<<p<<"% = C  Grade";
                else if(p>=50 && p<60)
                                cout<<p<<"% = D  Grade";
                else
                                cout<<p<<"% = F  Grade";
                getch();
}
```

Logical Operators

```
#include<conio.h>
void main(void)
    {   clrscr();
        char ch='y';
            while(!(ch=='n'))
            {
             cout<<"\nHello\n";
             cout<<"continue ";
              ch=getche();
            }
        cout<<"\nEnd";
        getch();
}
```

Output

```
Hello
continue a
Hello
continue y
Hello
continue k
Hello
continue n
End
```

Logical Operators

```
#include<iostream.h>
#include<conio.h>
void main(void)
{ clrscr();
      char ch='y';
            while(ch!='n')
            {
               cout<<"\nHello\n";
               cout<<“Do you want to continue \n";
                cout<<“Press y or n \n";
               ch=getche();

            }
      cout<<"\nEnd";getch();
      }
```

}

Logical Operators(Truth Table)

```
int main()
  {
   // create truth table for && (logical AND) operator
     cout << boolalpha << "Logical AND (&&)"
       << "\nfalse && false: " << ( false && false )
       << "\nfalse && true: " << ( false && true )
        << "\ntrue && false: " << ( true && false )
        << "\ntrue && true: " << ( true && true ) << "\n\n";
     // create truth table for || (logical OR) operator
     cout << "Logical OR (||)"
        << "\nfalse || false: " << ( false || false )
        << "\nfalse || true: " << ( false || true )
        << "\ntrue || false: " << ( true || false )
        << "\ntrue || true: " << ( true || true ) << "\n\n";
     // create truth table for ! (logical negation) operator
    cout << "Logical NOT (!)"
        << "\n!false: " << ( !false )
        << "\n!true: " << ( !true ) << endl;
     return 0; // indicate successful termination
 }
```

Output

```
Logical AND (&&)
          false && false: false
          false && true: false
          true && false: false
          true && true: true

Logical OR (||)
          false || false: false
          false || true: true
          true || false: true
          true || true: true

Logical NOT (!)
          !false: true
           !true: false
```

Operators

Operator Type	Operators
Unary	!, ++, --
Arithmetic	*, /, % +, -
Relational	<, >, <=, >= ==, !=
Logical	and && or ||
Conditional	? :
Assignment	=, +=, -=, *=, /=, %=

Bitwise Operators

- Works on bits (0/1) Binary values.
- Modify variables by considering the their bit pattern that represents the value stored.
- E.g.
  - int x=13; bit pattern for 13 is:0000 1101

Bitwise Operators

Operator	Symbol	Form	Operation
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
left shift	<<	x << y	all bits in x shifted left y bits
right shift	>>	x >> y	all bits in x shifted right y bits

```
#include <iostream>
main() {
        unsigned int a = 60; // 60 = 0011 1100
       unsigned int b = 13; // 13 = 0000 1101
        int c = 0;
        c = a & b; // 12 = 0000 1100
        cout << "Line 1 - Value of c is : " << c << endl ;
       c = a | b; // 61 = 0011 1101
       cout << "Line 2 - Value of c is: " << c << endl ;
        c = a ^ b; // 49 = 0011 0001
       cout << "Line 3 - Value of c is: " << c << endl ;
        c = ~a; // -61 = 1100 0011
       cout << "Line 4 - Value of c is: " << c << endl ;
       c = a << 2; // 240 = 1111 0000
       cout << "Line 5 - Value of c is: " << c << endl ;
        c = a >> 2; // 15 = 0000 1111
        cout << "Line 6 - Value of c is: " << c << endl ;
        return 0;
       }
```

Bitwise Operators

- When evaluating bitwise OR, if any bit in a column is 1, the result for that column is 1.
- When evaluating bitwise AND, if all bits in a column are 1, the result for that column is 1.
- When evaluating bitwise XOR, if there are an odd number of 1 bits in a column, the result for that column is 1.