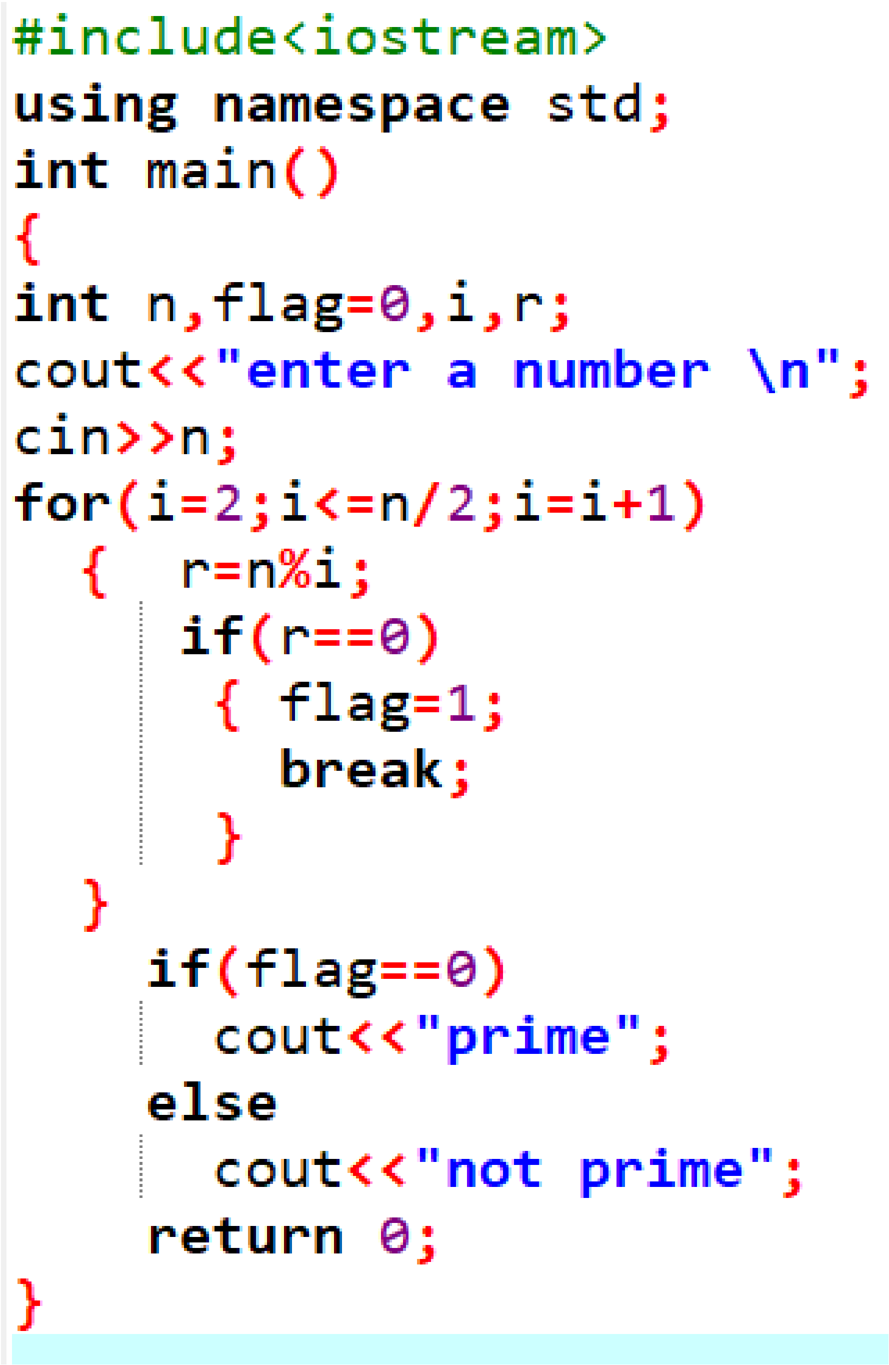
MA144: Problem Solving and Computer Programming

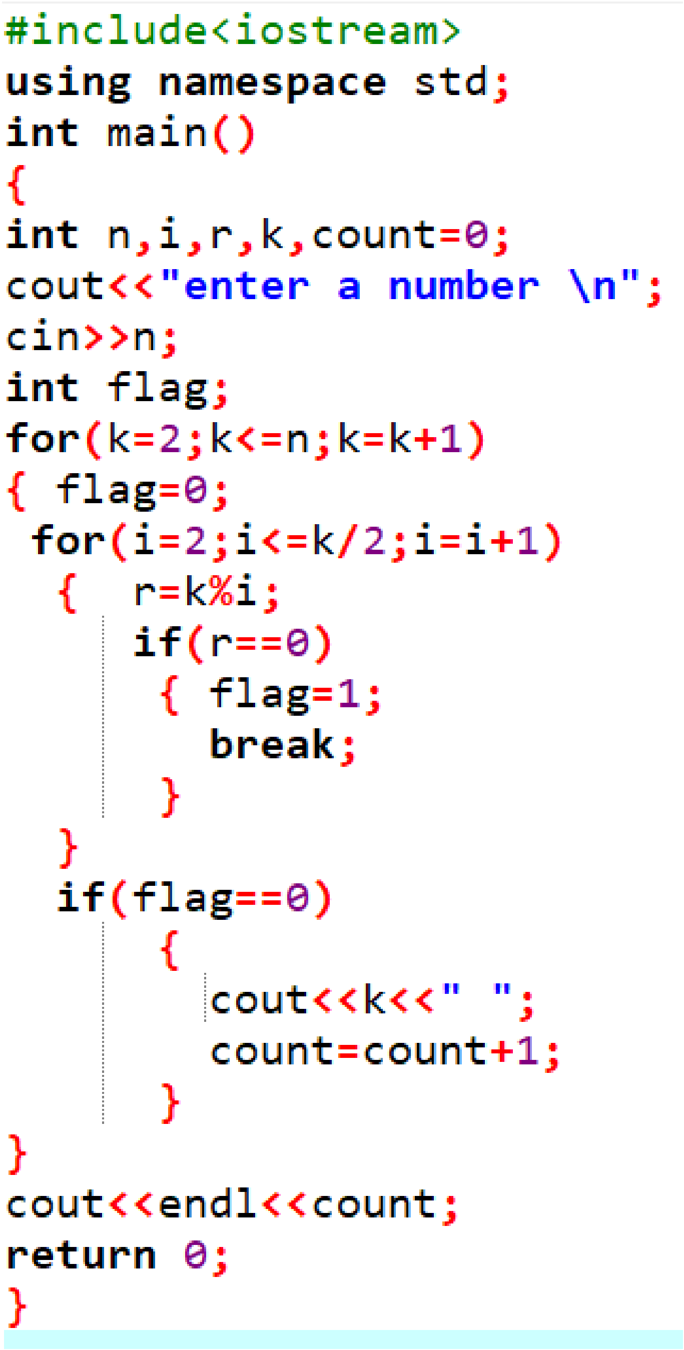
Lecture-9

Operators

Program for determining prime or not



```
#include<iostream>
using namespace std;
int main()
{
int n,flag=0,i,r;
cout<<"enter a number \n";
cin>>n;
for(i=2;i<=n/2;i=i+1)
  {  r=n%i;
     if(r==0)
      { flag=1;
        break;
      }
  }
    if(flag==0)
      cout<<"prime";
    else
      cout<<"not prime";
    return 0;
}
```



```cpp
#include<iostream>
using namespace std;
int main()
{
int n,i,r,k,count=0;
cout<<"enter a number \n";
cin>>n;
int flag;
for(k=2;k<=n;k=k+1)
{ flag=0;
 for(i=2;i<=k/2;i=i+1)
  {  r=k%i;
     if(r==0)
      { flag=1;
        break;
      }
  }
  if(flag==0)
      {
        cout<<k<<" ";
        count=count+1;
      }
}
cout<<endl<<count;
return 0;
}
```

Increment and Decrement Operators

Increment Operators (++) *unary operators*

- **postfix** (placed after the variable)

a++

- **prefix** (placed before the variable)

++a

Decrement Operators (--) *unary operators*

- **postfix** (placed after the variable)

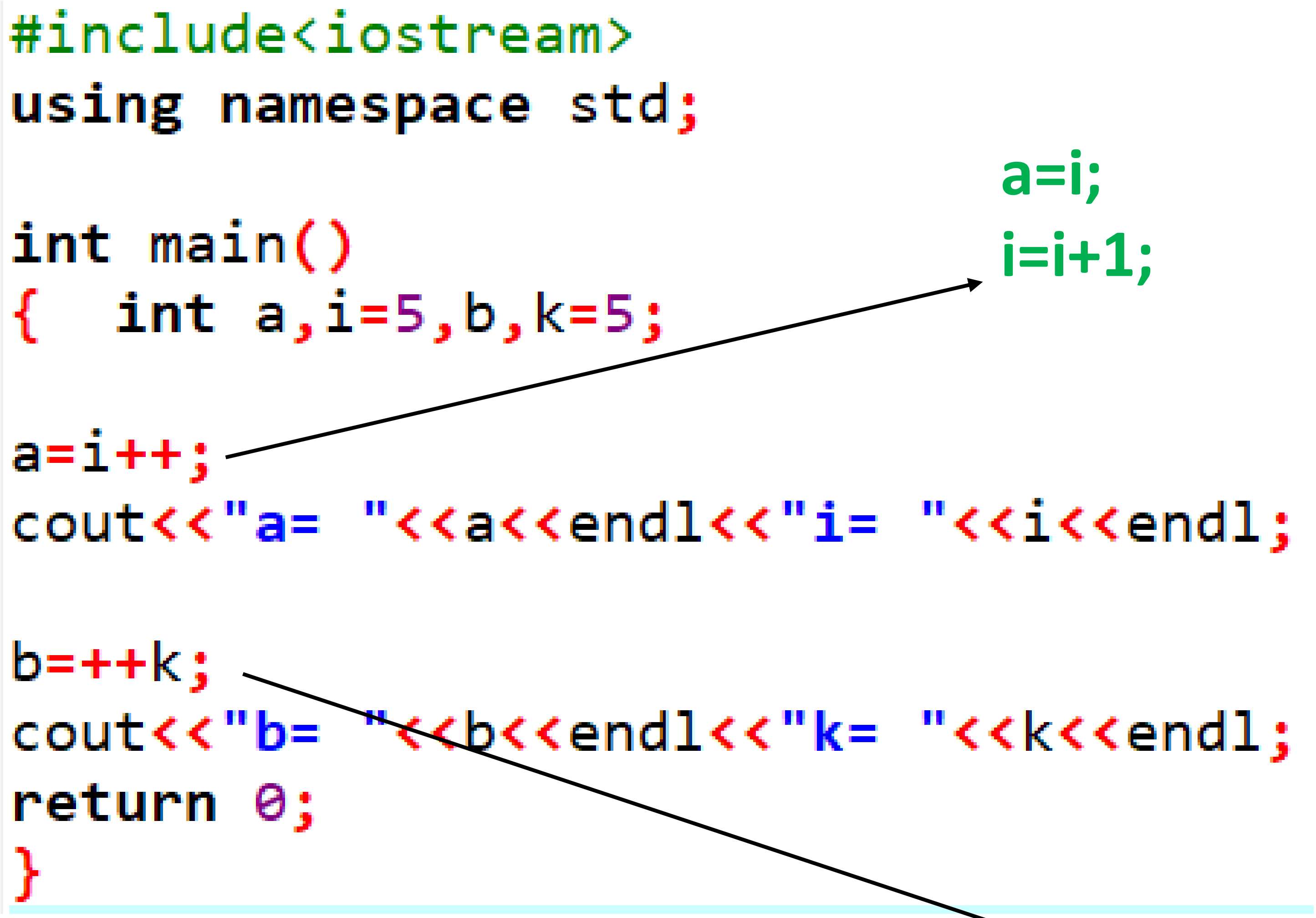
a--

- **prefix** (placed before the variable)

--a

Postfix: left-right **Prefix: right-left**

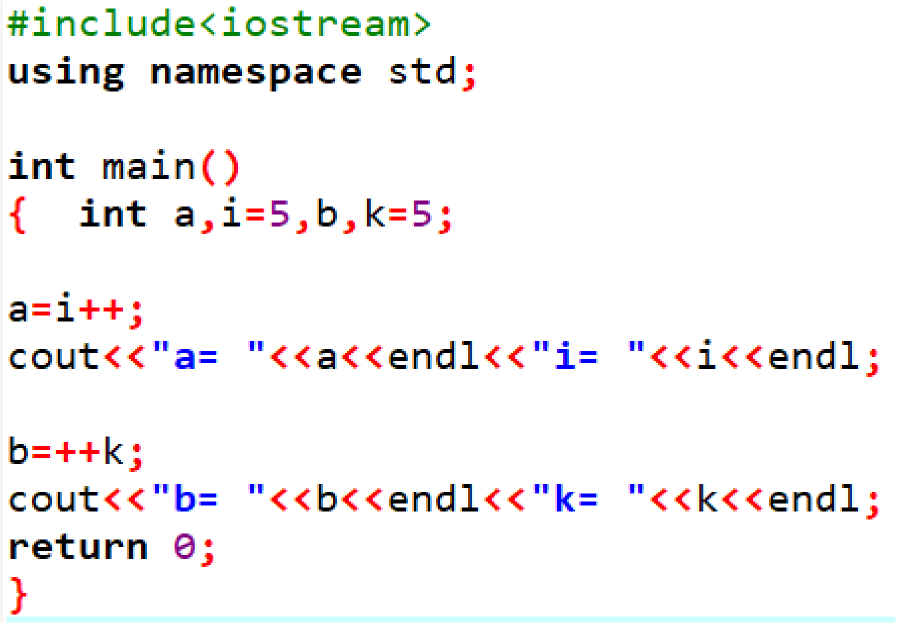
:: scope resolution operator	*Highest precedence (done first)*
. dot operator -> member selection [] array indexing ( ) function call ++ postfix increment operator (placed after the variable) -- postfix decrement operator (placed after the variable)	
++ prefix increment operator (placed before the variable) -- prefix decrement operator (placed before the variable) ! not - unary minus + unary plus * dereference & address of *new* *delete* *delete[]* *sizeof*	
* multiplication / division % remainder (modulo)	
+ addition - subtraction	
<< insertion operator (output) >> extraction operator (input)	
< less than <= less than or equal > greater than >= greater than or equal	
== equal != not equal	
&& and	
|| or	
= assignment += add and assign -= subtract and assign *= multiply and assign /= divide and assign %= modulo and assign	*Lowest precedence (done last)*



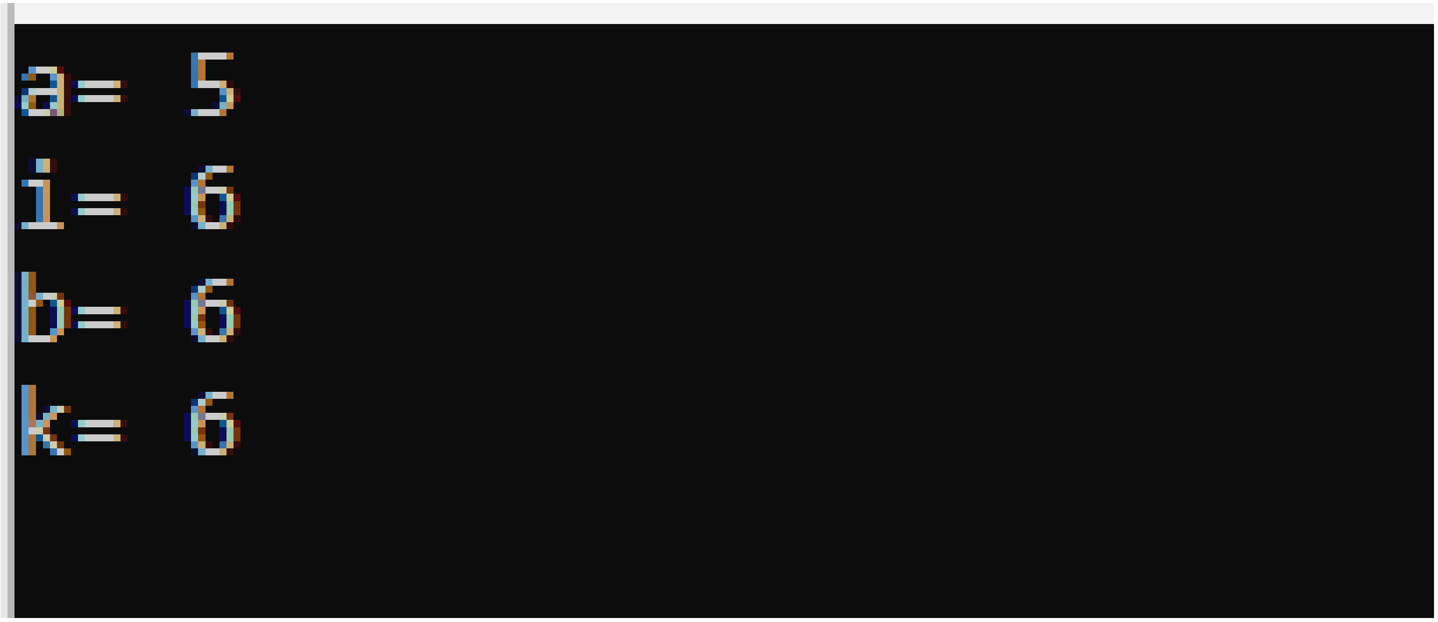
```
#include<iostream>
using namespace std;

int main()
{   int a,i=5,b,k=5;

a=i++;
cout<<"a= "<<a<<endl<<"i= "<<i<<endl;

b=++k;
cout<<"b= "<<b<<endl<<"k= "<<k<<endl;
return 0;
}
```

a=i;
i=i+1;

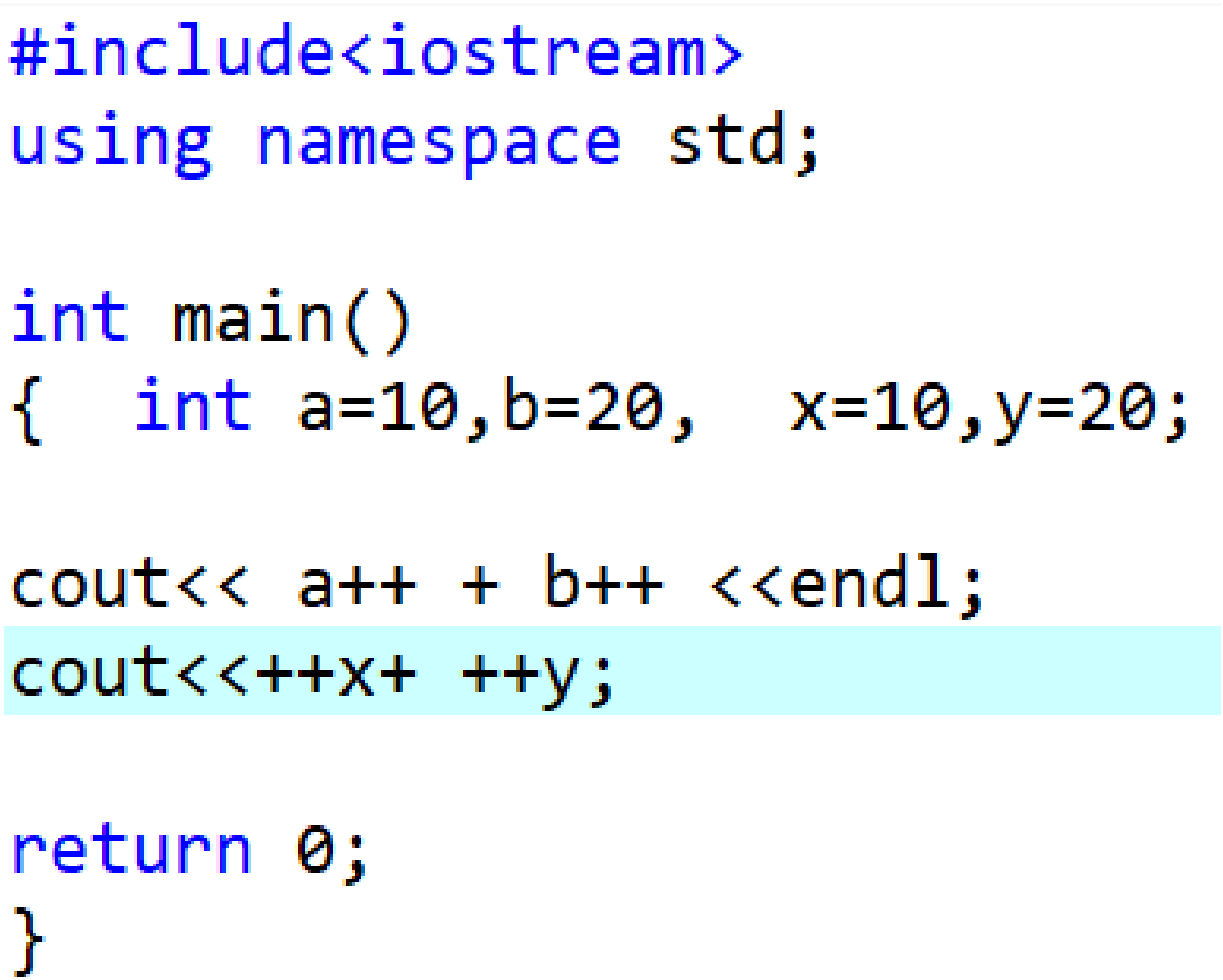
k=k+1;
b=k;



```cpp
#include<iostream>
using namespace std;

int main()
{   int a,i=5,b,k=5;

a=i++;
cout<<"a= "<<a<<endl<<"i= "<<i<<endl;

b=++k;
cout<<"b= "<<b<<endl<<"k= "<<k<<endl;
return 0;
}
```

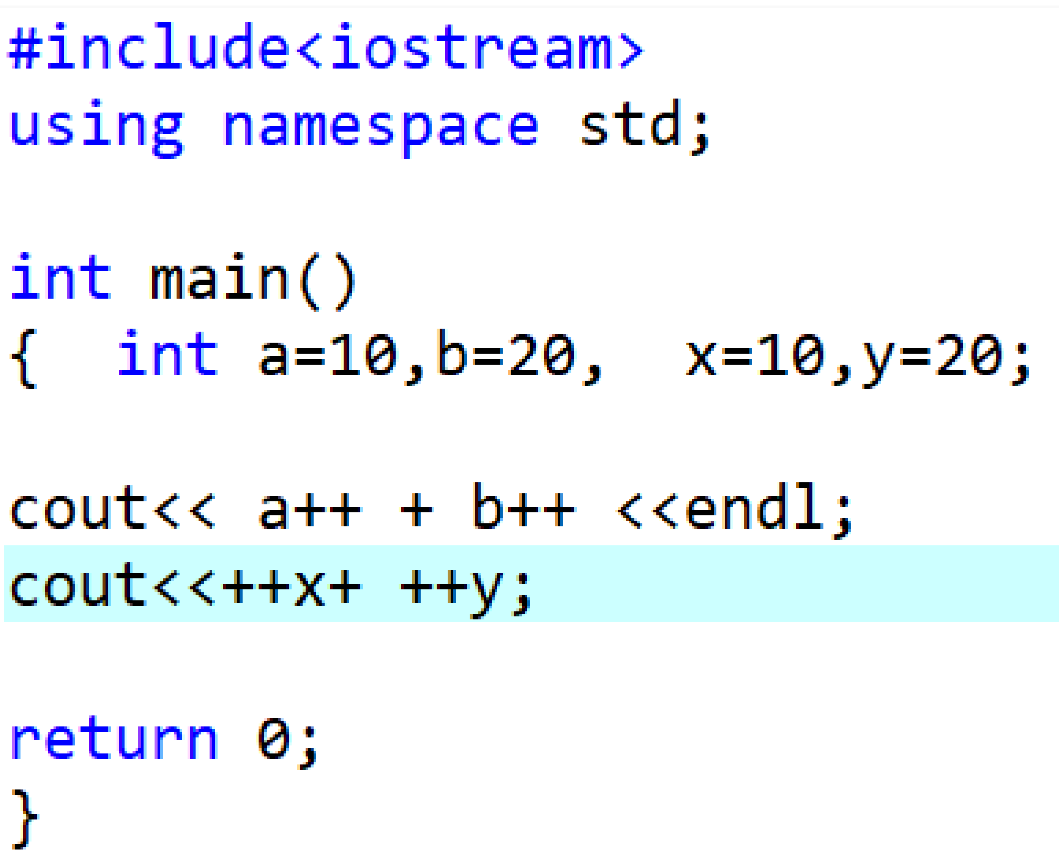


```
a=  5
i=  6
b=  6
k=  6
```

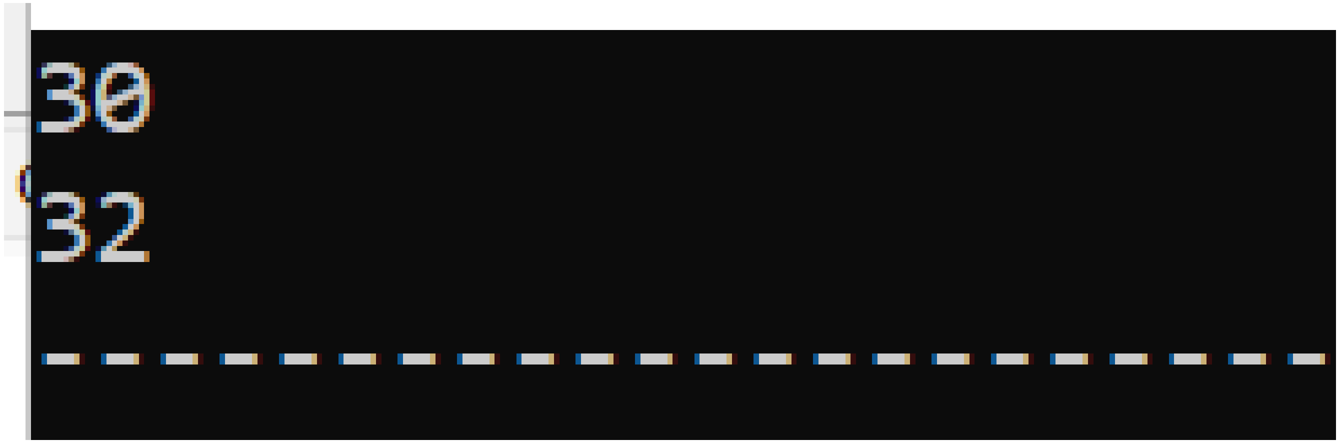


```
#include<iostream>
using namespace std;

int main()
{  int a=10,b=20,  x=10,y=20;

cout<< a++ + b++ <<endl;
cout<<++x+ ++y;

return 0;
}
```

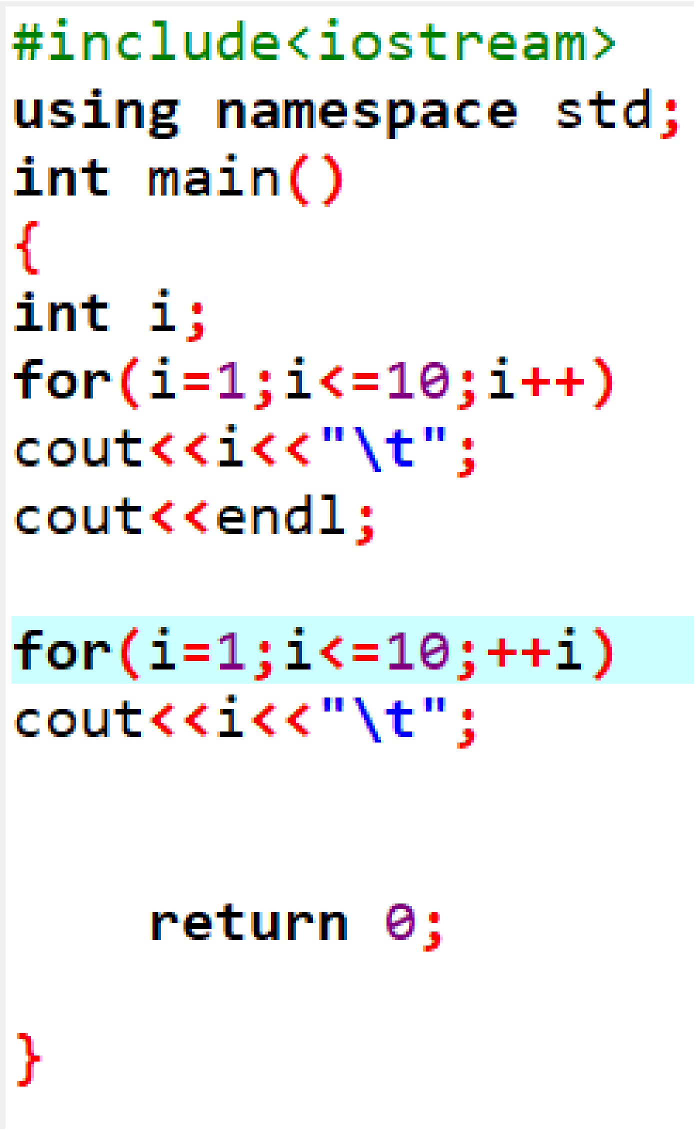
What is output?



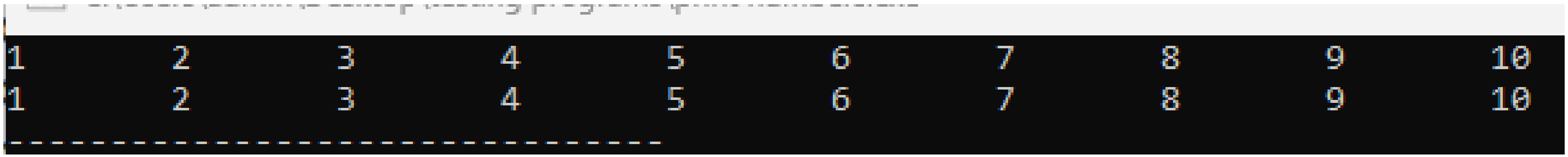
```
#include<iostream>
using namespace std;

int main()
{  int a=10,b=20,  x=10,y=20;

cout<< a++ + b++ <<endl;
cout<<++x+ ++y;

return 0;
}
```



```
30
32

----------------------
```



```
#include<iostream>
using namespace std;
int main()
{
int i;
for(i=1;i<=10;i++)
cout<<i<<"\t";
cout<<endl;

for(i=1;i<=10;++i)
cout<<i<<"\t";



    return 0;

}
```



```
1       2       3       4       5       6       7       8       9       10
1       2       3       4       5       6       7       8       9       10
--------------------------------
```

Logical Operators (binary operators)

Logical AND &&

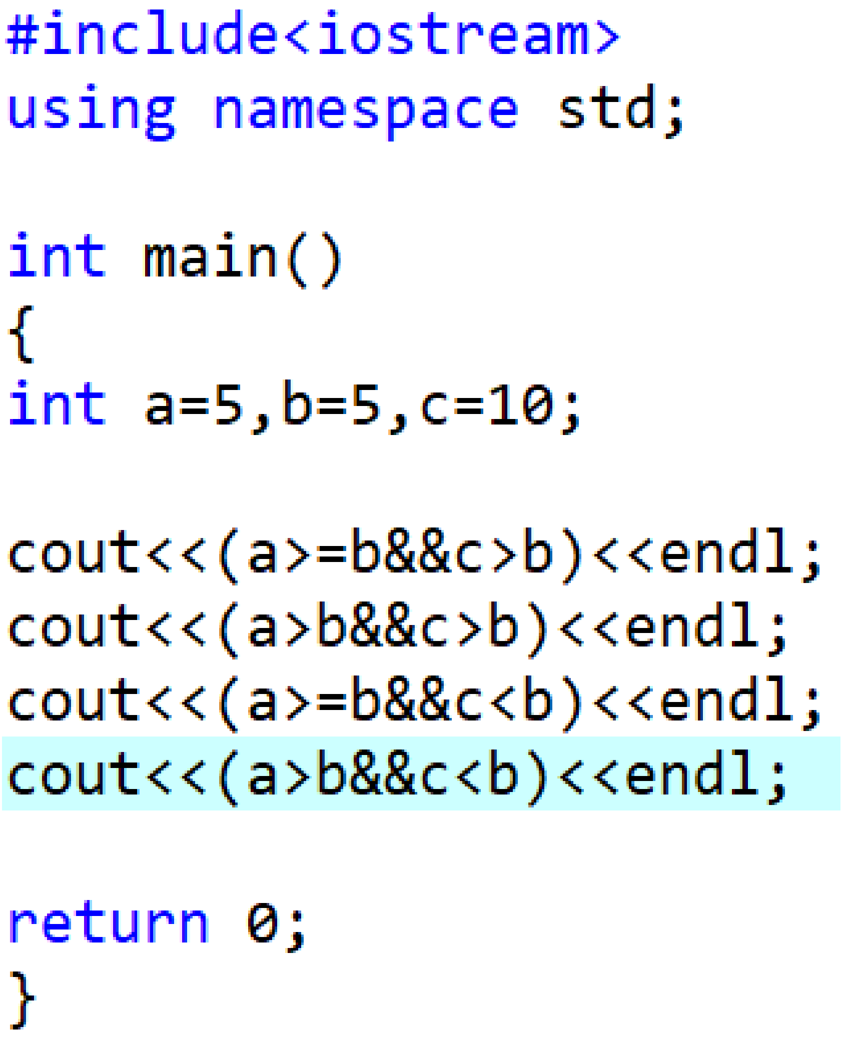
0 && 0 = 0

0 && 1 = 0

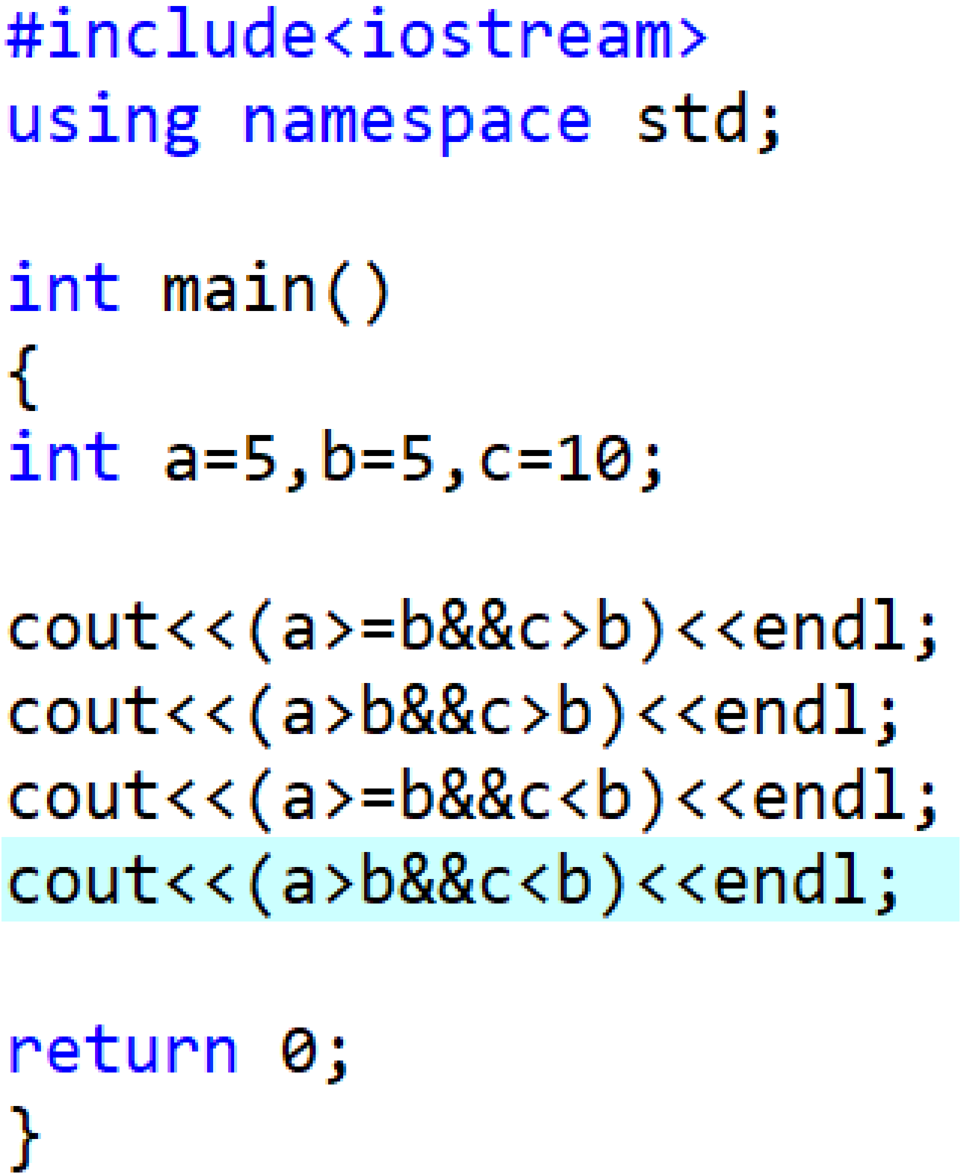
1 && 0 = 0

1 && 1 = 1

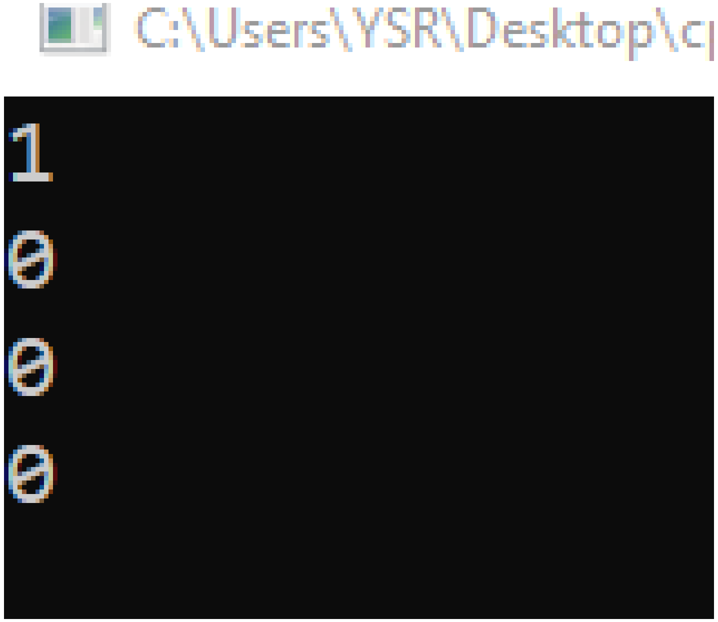
Left-right associativity



```
#include<iostream>
using namespace std;

int main()
{
int a=5,b=5,c=10;

cout<<(a>=b&&c>b)<<endl;
cout<<(a>b&&c>b)<<endl;
cout<<(a>=b&&c<b)<<endl;
cout<<(a>b&&c<b)<<endl;

return 0;
}
```

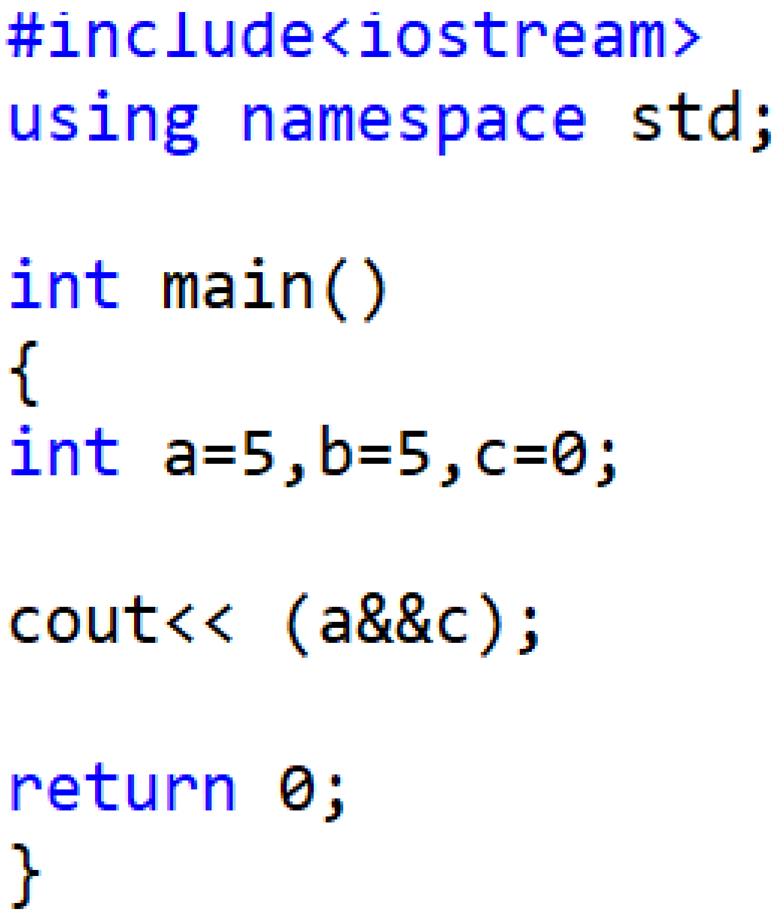


```
#include<iostream>
using namespace std;

int main()
{
int a=5,b=5,c=10;

cout<<(a>=b&&c>b)<<endl;
cout<<(a>b&&c>b)<<endl;
cout<<(a>=b&&c<b)<<endl;
cout<<(a>b&&c<b)<<endl;

return 0;
}
```



C:\Users\YSR\Desktop\c

```
1
0
0
0
```



```
#include<iostream>
using namespace std;

int main()
{
int a=5,b=5,c=0;

cout<< (a&&c);

return 0;
}
```

What is output?

How to represent $2< x< 3$ in C++?

Logical OR ||

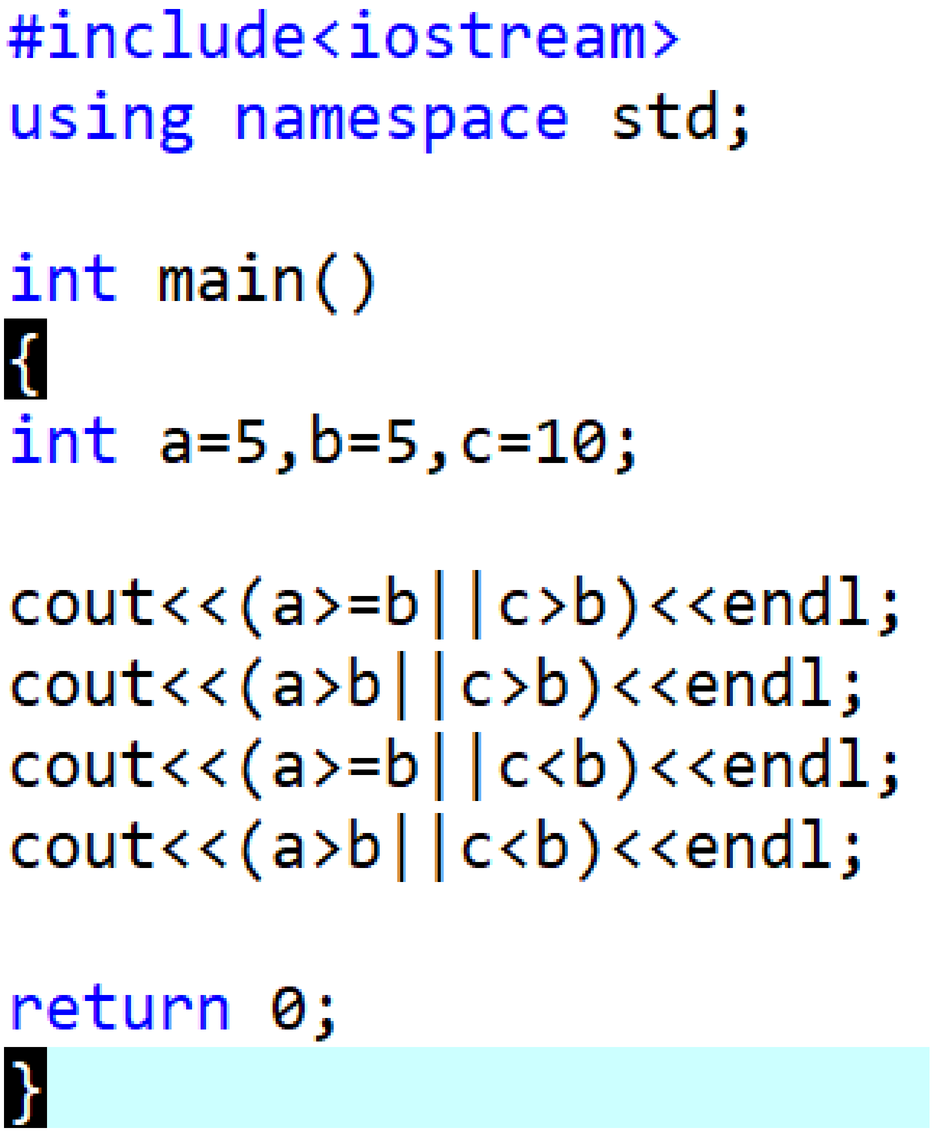
0 || 0 = 0

0 || 1 = 1

1 || 0 = 1

1 || 1 = 1

Left-right associativity



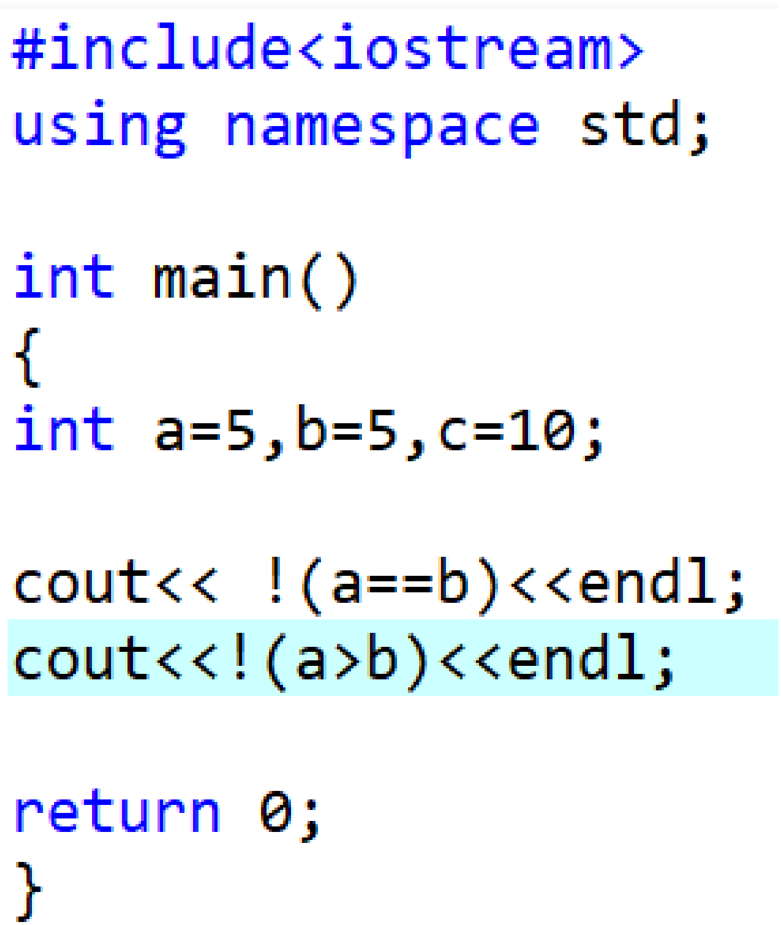
```
#include<iostream>
using namespace std;

int main()
{
int a=5,b=5,c=10;

cout<<(a>=b||c>b)<<endl;
cout<<(a>b||c>b)<<endl;
cout<<(a>=b||c<b)<<endl;
cout<<(a>b||c<b)<<endl;

return 0;
}
```

Logical NOT !

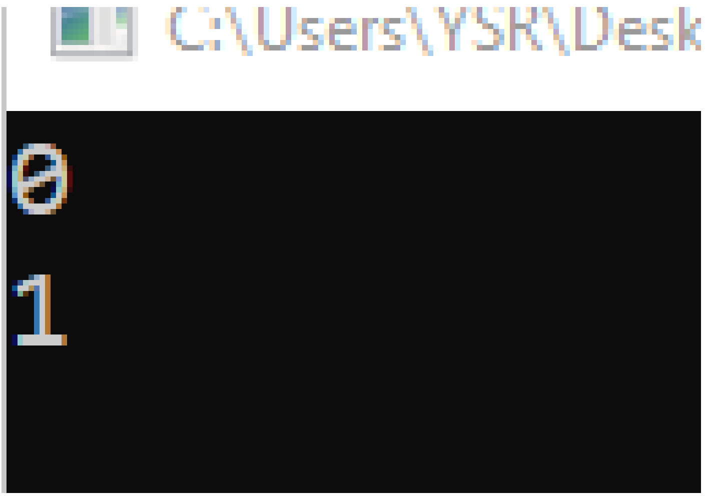
(unary operator)

! 1 = 0

! 0 = 1



```
#include<iostream>
using namespace std;

int main()
{
int a=5,b=5,c=10;

cout<< !(a==b)<<endl;
cout<<!(a>b)<<endl;

return 0;
}
```

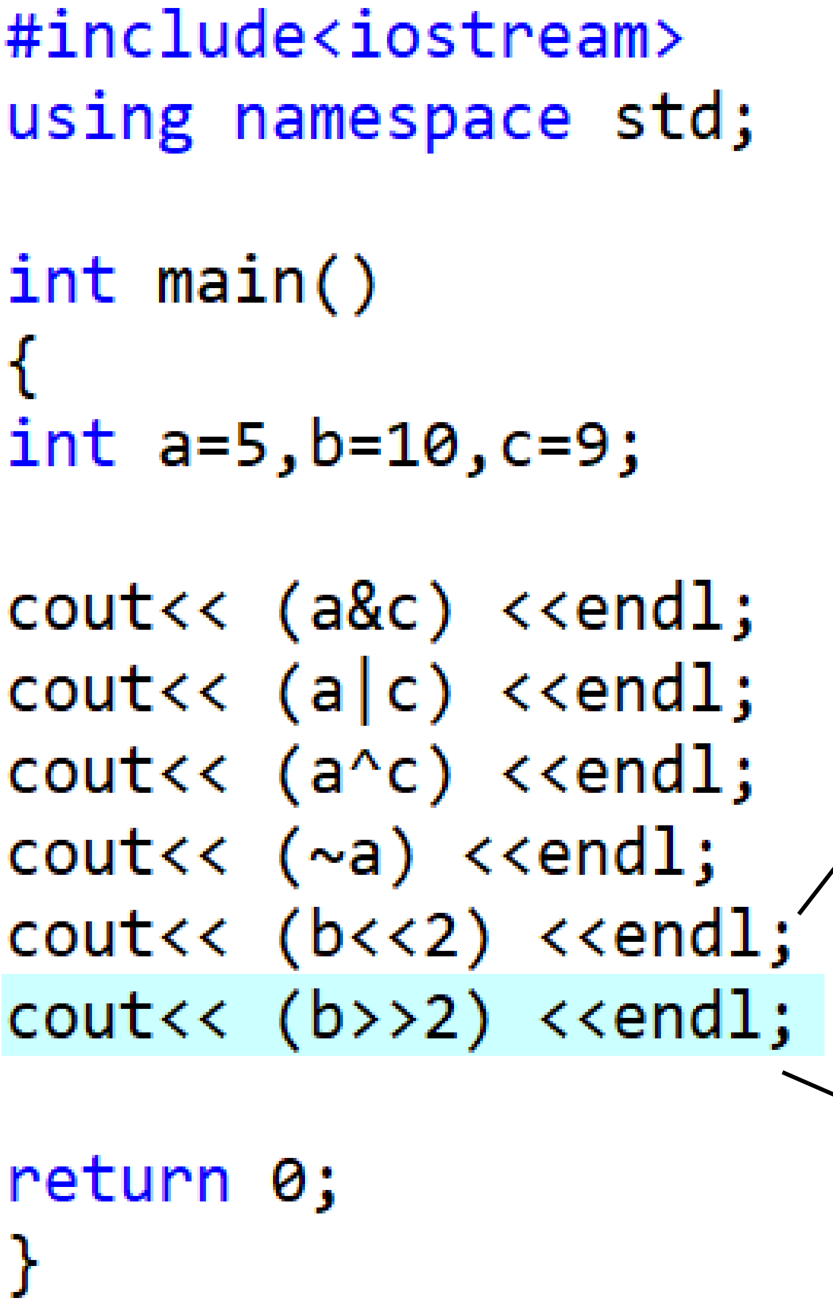


```
0
1
```

Bit-Wise Operators

The bitwise operators are used for bit manipulation;
these may only be applied to unsigned integers

- & bitwise AND
- | bitwise inclusive OR
- ^ bitwise exclusive OR
- << left shift
  (shifts the specified number of bits to left)
- >> right shift
  (shifts specified number of bits to right)
- ~ one’s complement (unary)
  (complements al1 1’s to 0’s and all 0’s to 1’s)

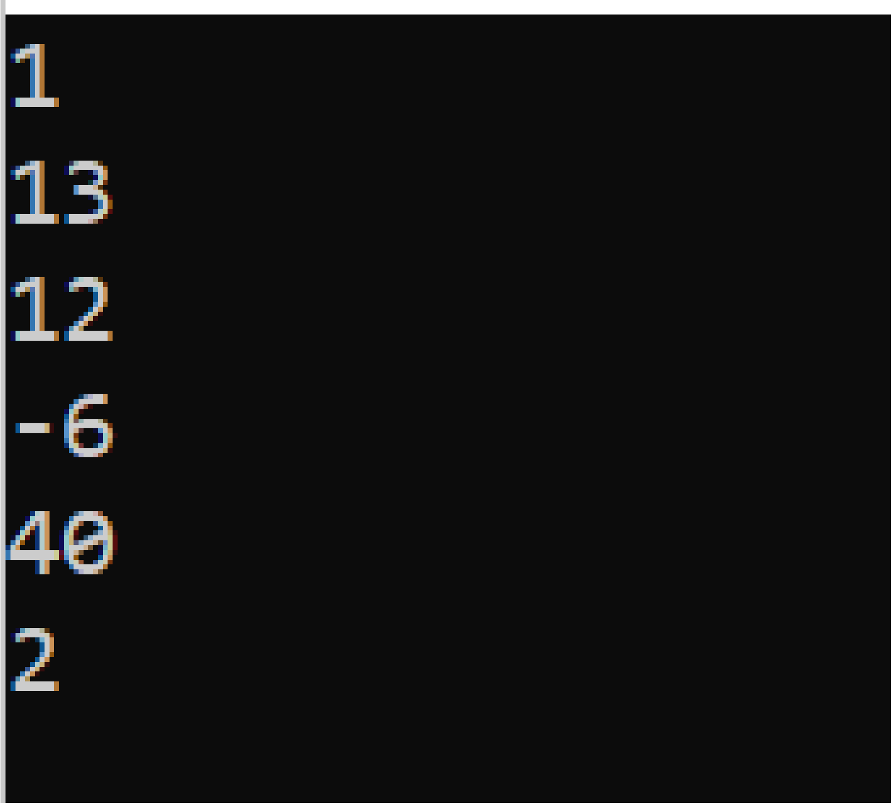


```
#include<iostream>
using namespace std;

int main()
{
int a=5,b=10,c=9;

cout<< (a&c) <<endl;
cout<< (a|c) <<endl;
cout<< (a^c) <<endl;
cout<< (~a) <<endl;
cout<< (b<<2) <<endl;
cout<< (b>>2) <<endl;

return 0;
}
```

moves 2 bits to left

moves 2 bits to right

Output



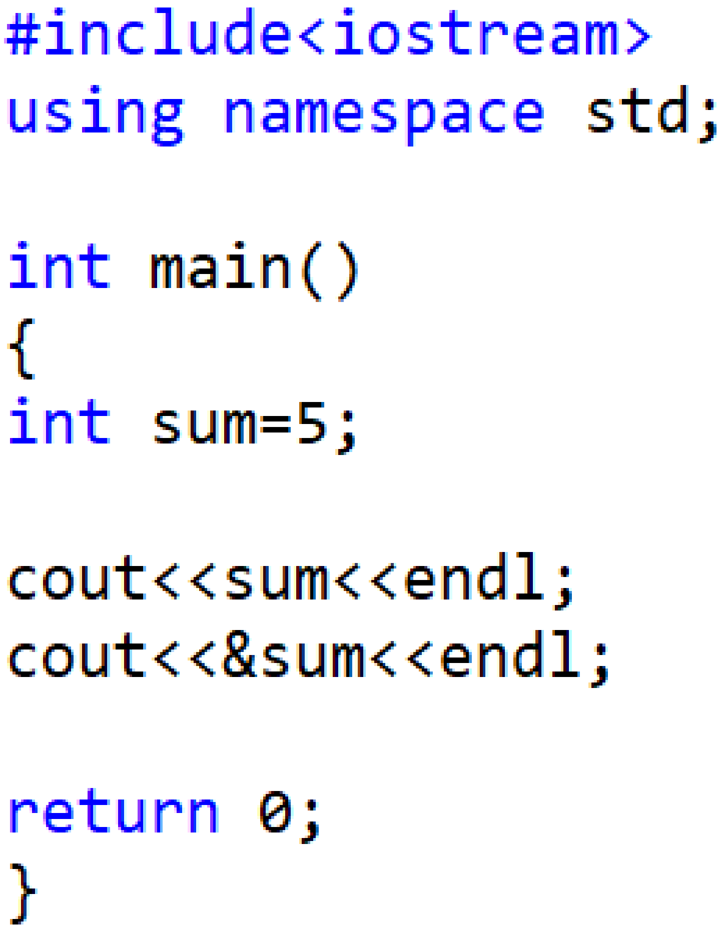
```
1
13
12
-6
40
2
```

address of operator &

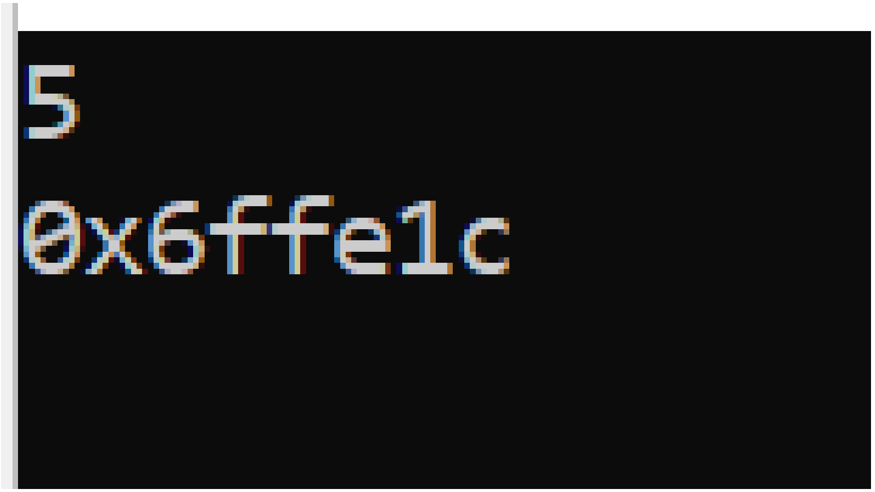
Syntax

& var

& in front of an *ordinary* variable produces the address of that variable



```
#include<iostream>
using namespace std;

int main()
{
int sum=5;

cout<<sum<<endl;
cout<<&sum<<endl;

return 0;
}
```



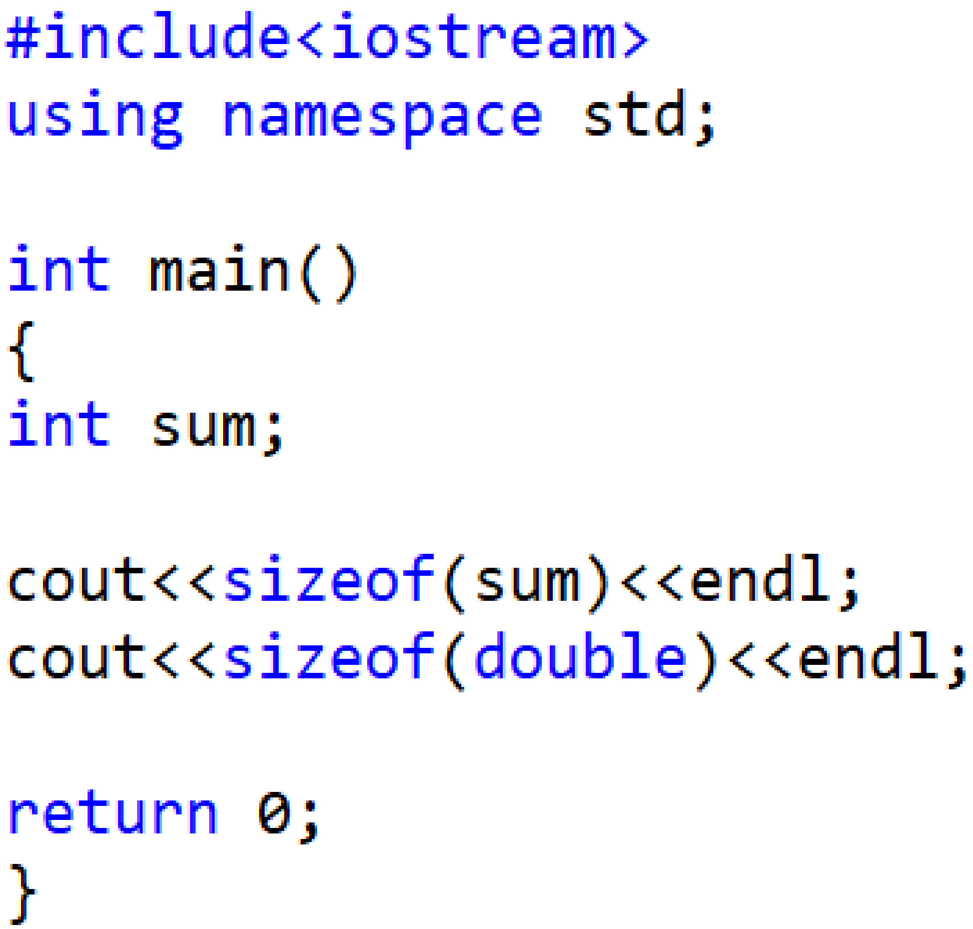
```
5
0x6ffe1c
```

sizeof operator

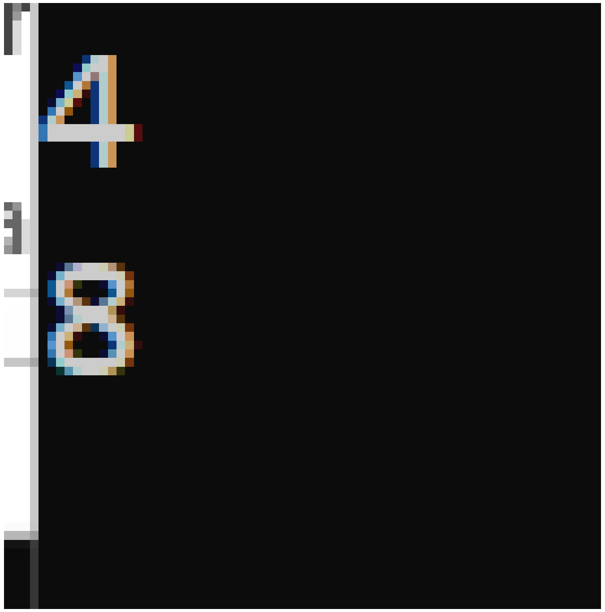
It determines the size, in bytes, of a variable or data type.

Syntax

```
sizeof (variable)

sizeof (data_type)
```



```
#include<iostream>
using namespace std;

int main()
{
int sum;

cout<<sizeof(sum)<<endl;
cout<<sizeof(double)<<endl;

return 0;
}
```



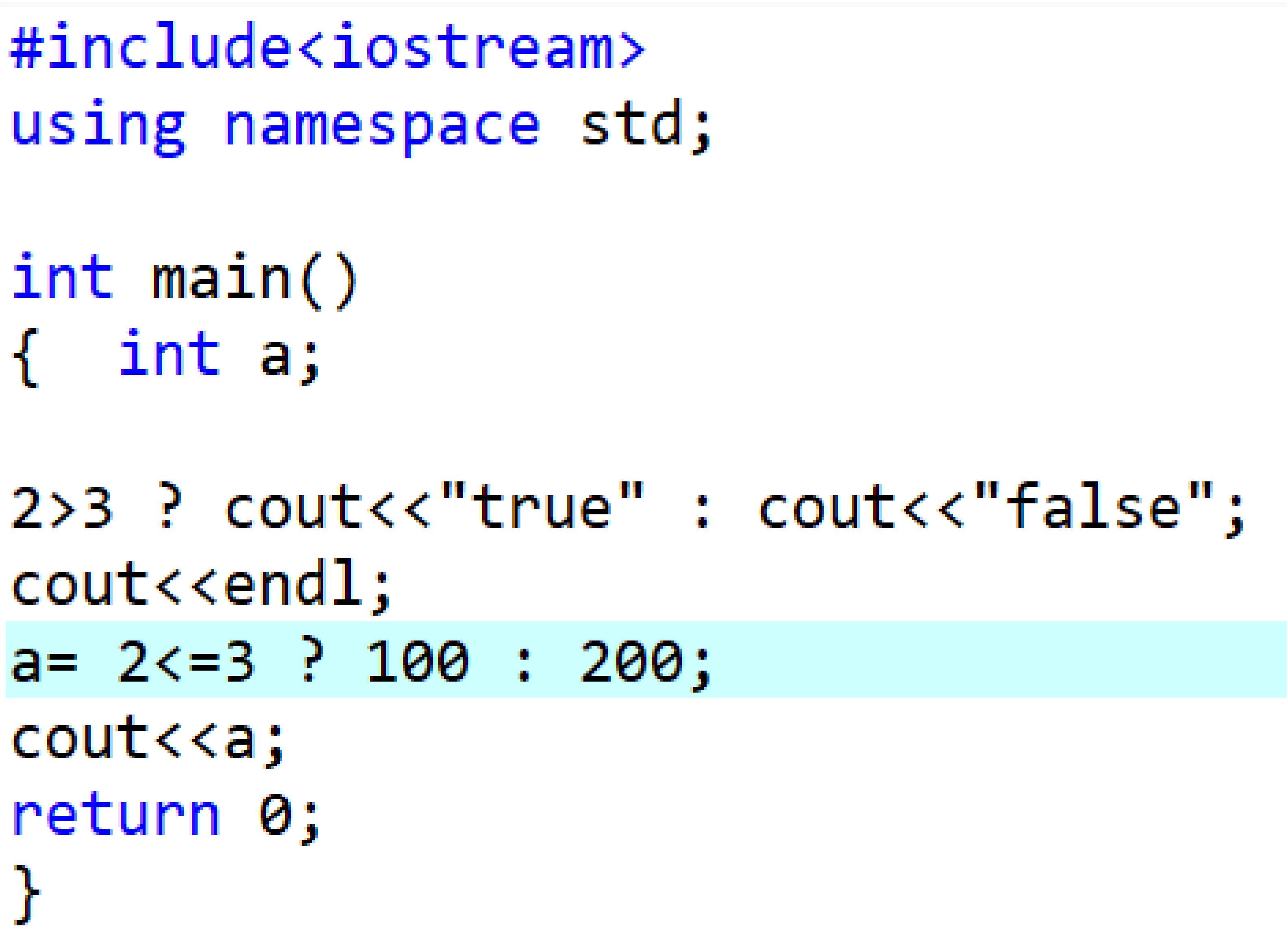
```
4
8
```

Ternary (or conditional) operator ? :

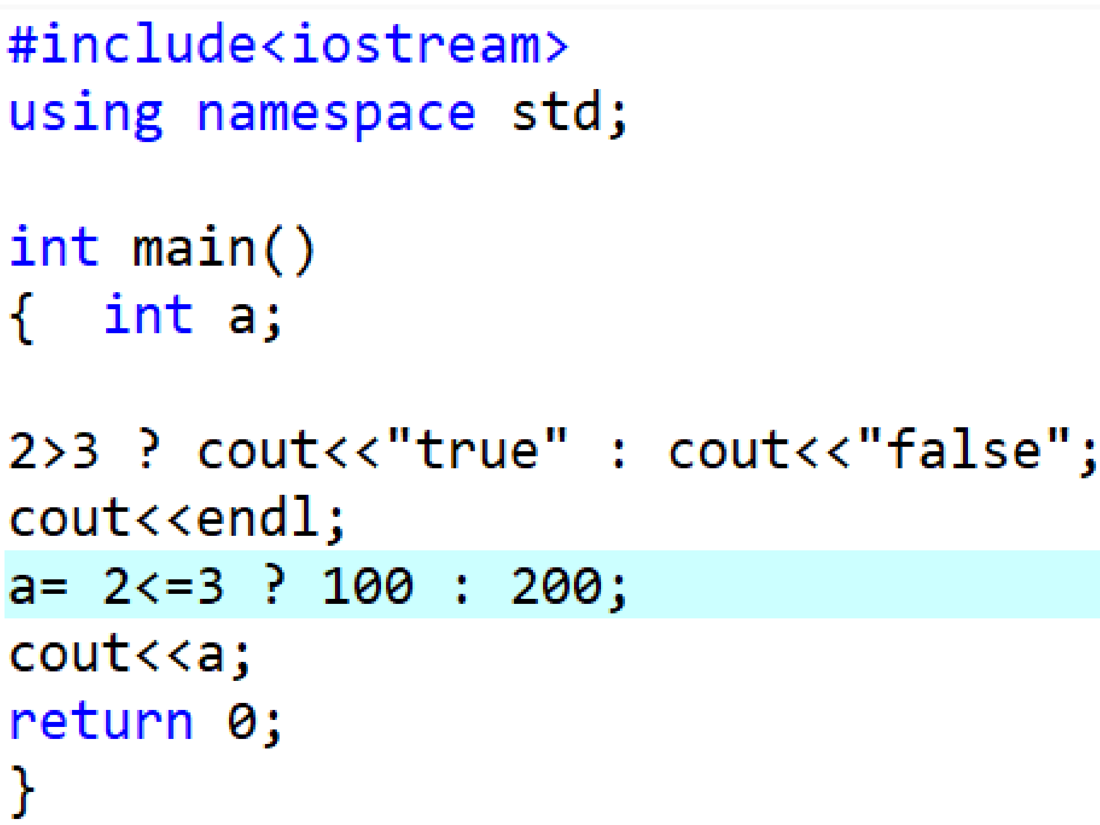
```
condition ? expression_1 : expression_2;
```

Why ternary

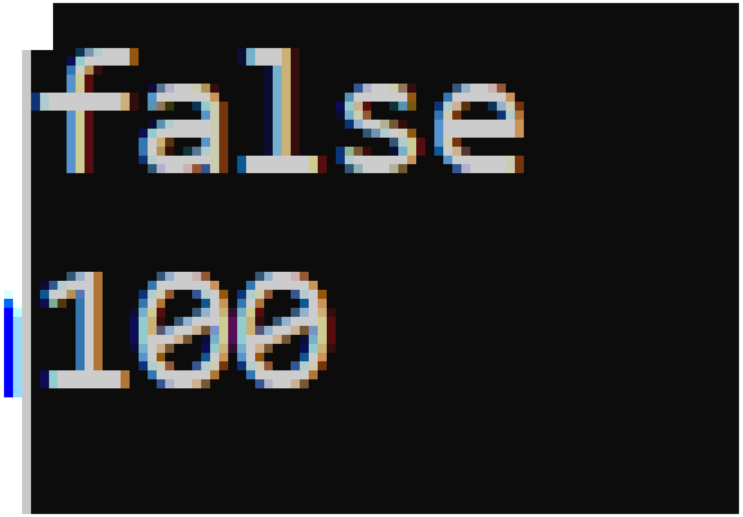
- If `condition` is true, `expression_1` is evaluated
- If `condition` is false, `expression_2` is evaluated



```
#include<iostream>
using namespace std;

int main()
{  int a;

2>3 ? cout<<"true" : cout<<"false";
cout<<endl;
a= 2<=3 ? 100 : 200;
cout<<a;
return 0;
}
```



```
#include<iostream>
using namespace std;

int main()
{  int a;

2>3 ? cout<<"true" : cout<<"false";
cout<<endl;
a= 2<=3 ? 100 : 200;
cout<<a;
return 0;
}
```



```
false
100
```