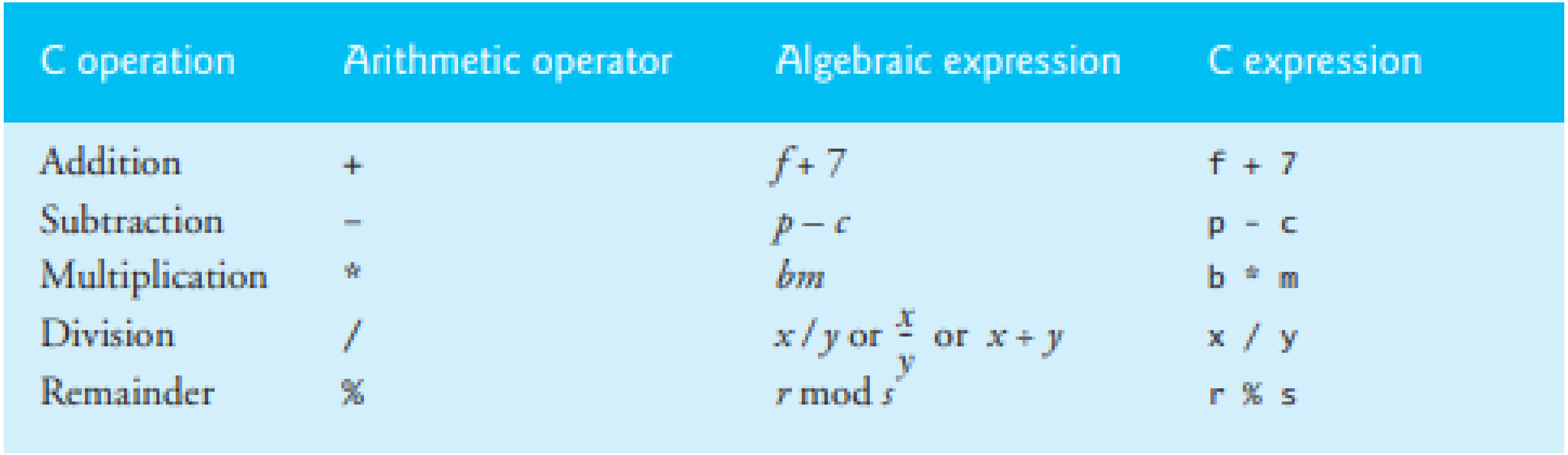
19 CSE 102 Computer Programming

- Operators and Decision Making
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Operator Precedence
  - If Else
  - Conditional Operators

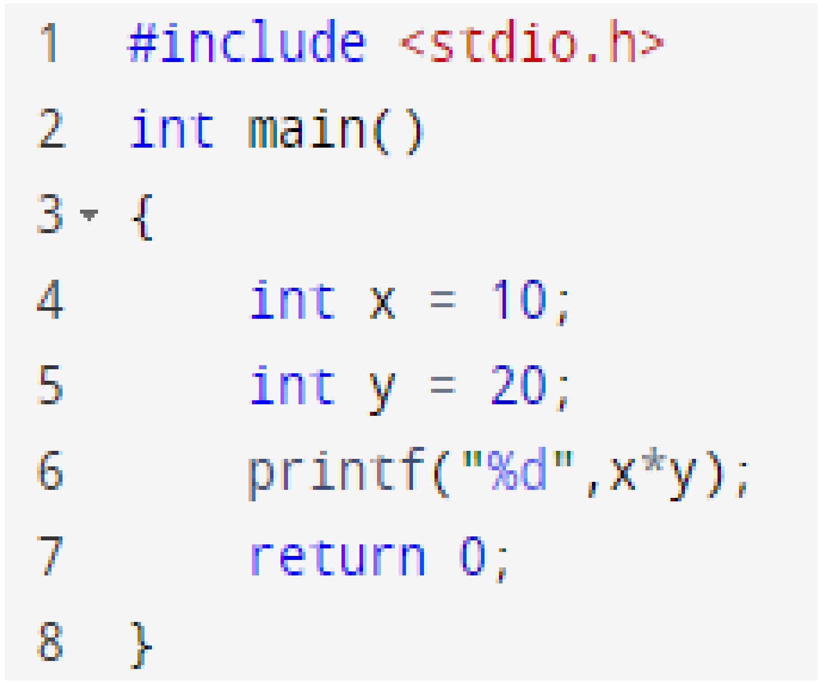
Arithmetic operators



C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Referred from : Deitel and Deitel – How to Program in C

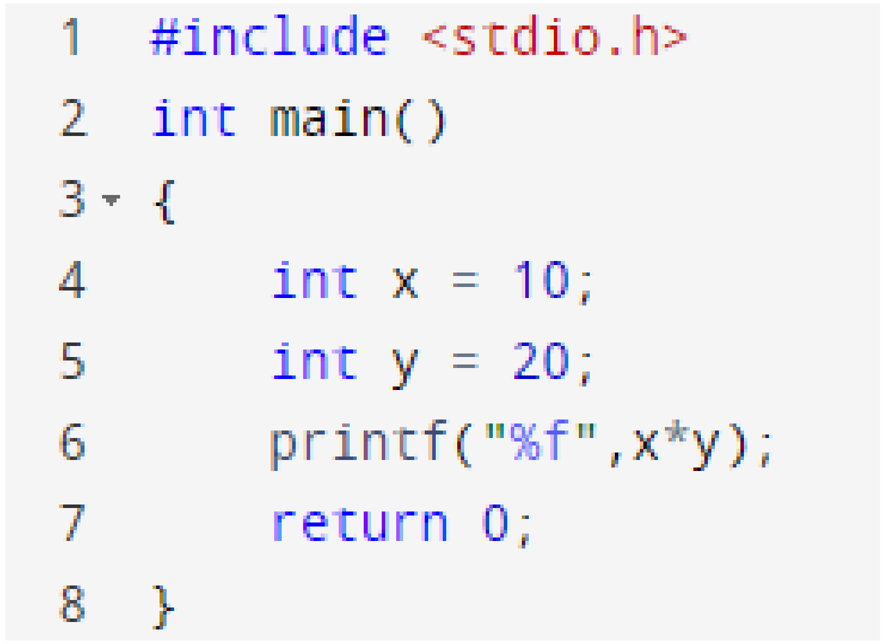
Multiplication operator 1



```c
#include <stdio.h>
int main()
{
      int x = 10;
      int y = 20;
      printf("%d",x*y);
      return 0;
}
```

200

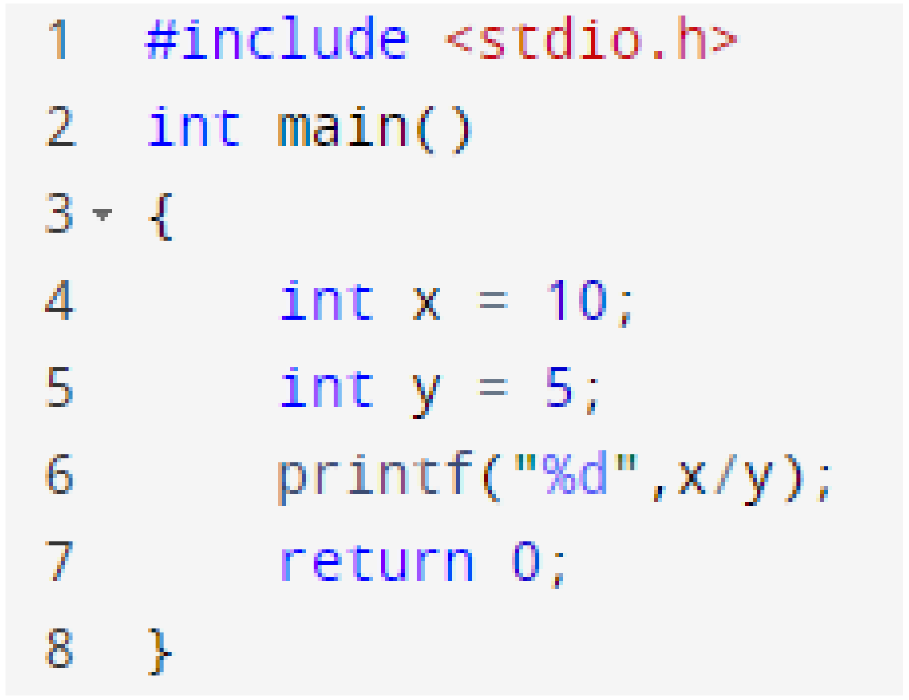
0.000



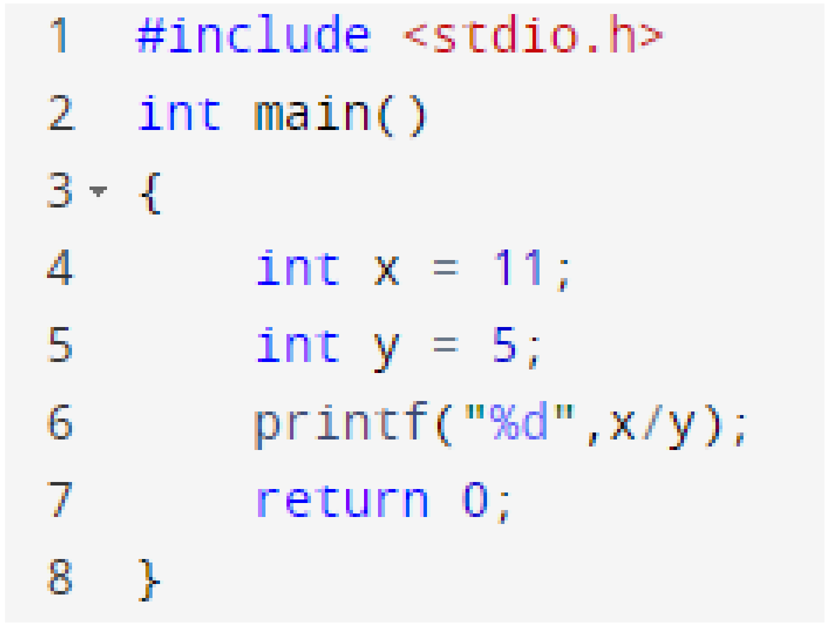
```c
#include <stdio.h>
int main()
{
      int x = 10;
      int y = 20;
      printf("%f",x*y);
      return 0;
}
```

- Study the code
- Run the two programs
- **Why did the program on the right side give an output of 0.000?**

Division operator 1



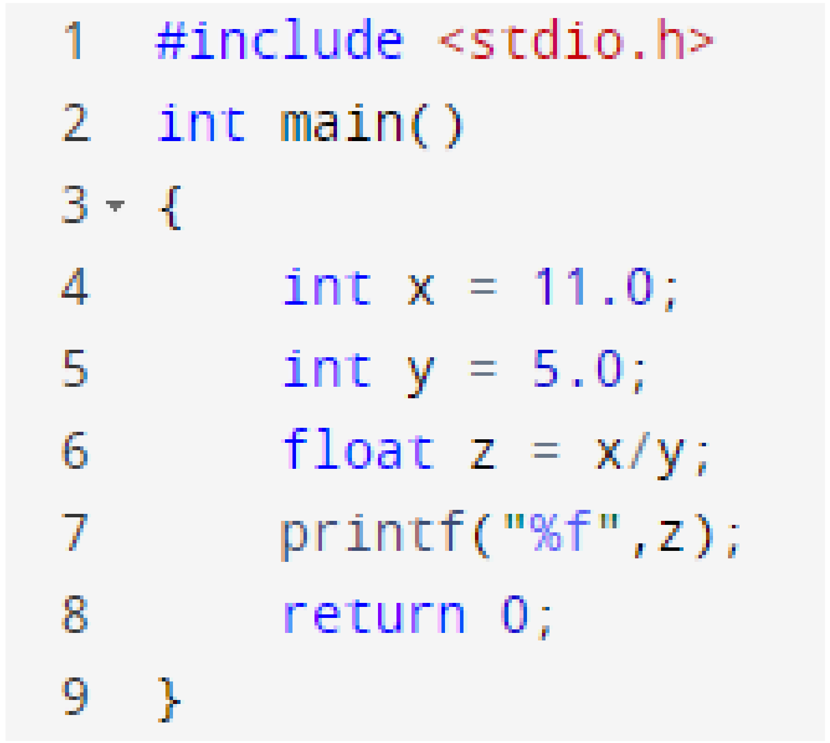
```c
#include <stdio.h>
int main()
{
	int x = 10;
	int y = 5;
	printf("%d",x/y);
	return 0;
}
```



```c
#include <stdio.h>
int main()
{
	int x = 11;
	int y = 5;
	printf("%d",x/y);
	return 0;
}
```

- Study the code
- Run the two programs
- Note the output of / operator with respect to division by integers

Division operator 2



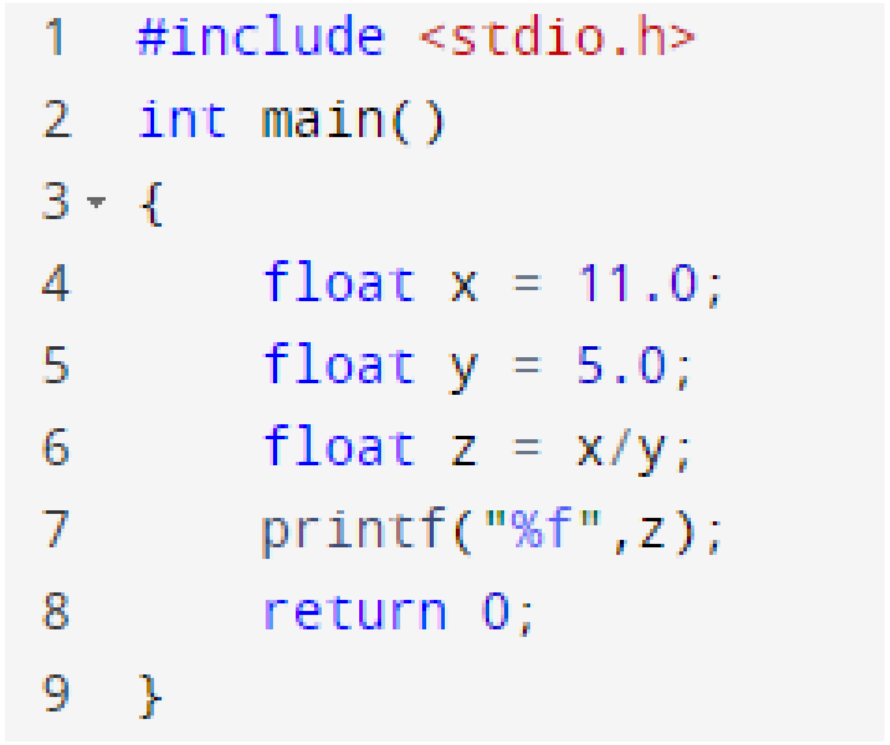
```c
#include <stdio.h>
int main()
{
        int x = 11.0;
        int y = 5.0;
        float z = x/y;
        printf("%f",z);
        return 0;
}
```

2.00

Output

2.200

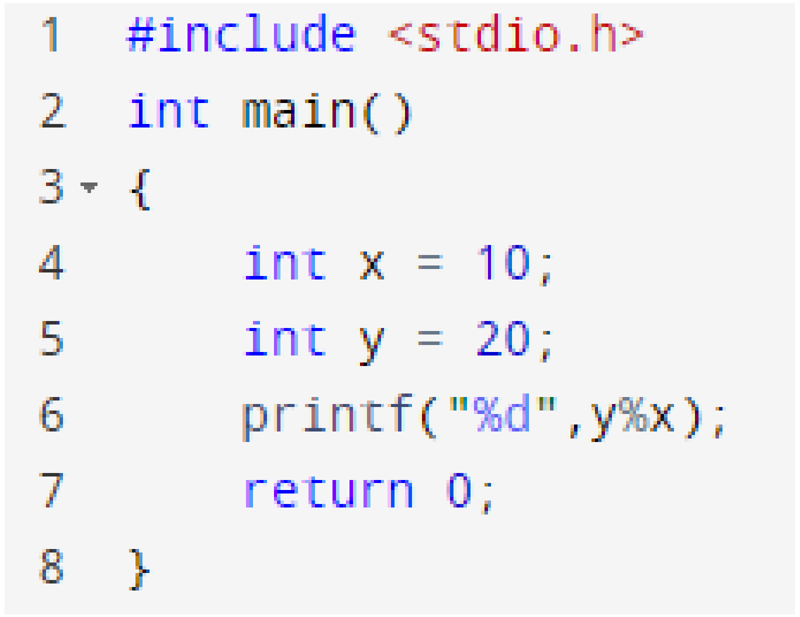
Output



```c
#include <stdio.h>
int main()
{
        float x = 11.0;
        float y = 5.0;
        float z = x/y;
        printf("%f",z);
        return 0;
}
```

- Study the code
- Run the two programs
- **Note the output of / operator with respect to division by integers and floating-point numbers**

Modulus operator 1



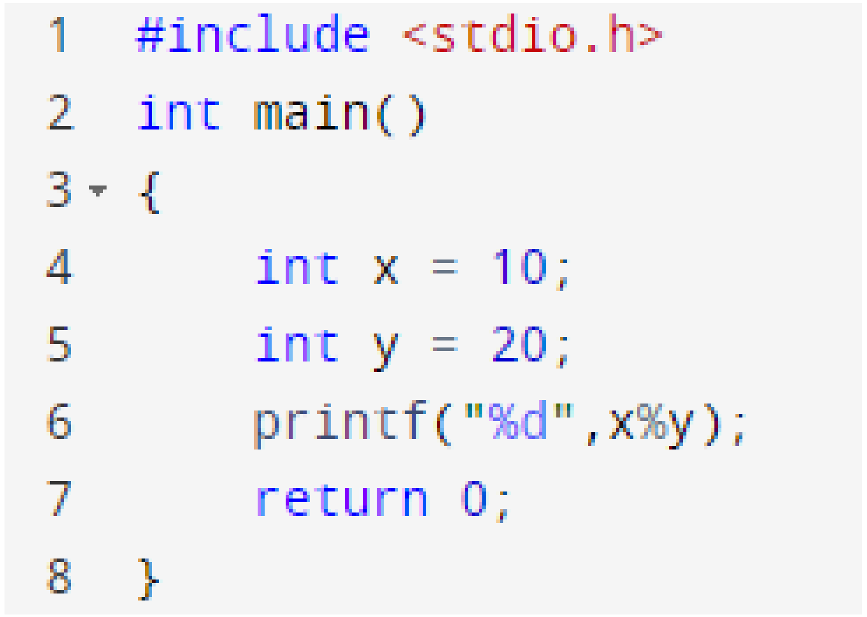
```c
#include <stdio.h>
int main()
{
        int x = 10;
        int y = 20;
        printf("%d",y%x);
        return 0;
}
```

0

Output

10

Output



```c
#include <stdio.h>
int main()
{
        int x = 10;
        int y = 20;
        printf("%d",x%y);
        return 0;
}
```

- Study the code
- Run the two programs
- **Note the output of % operator for the program on the right side**

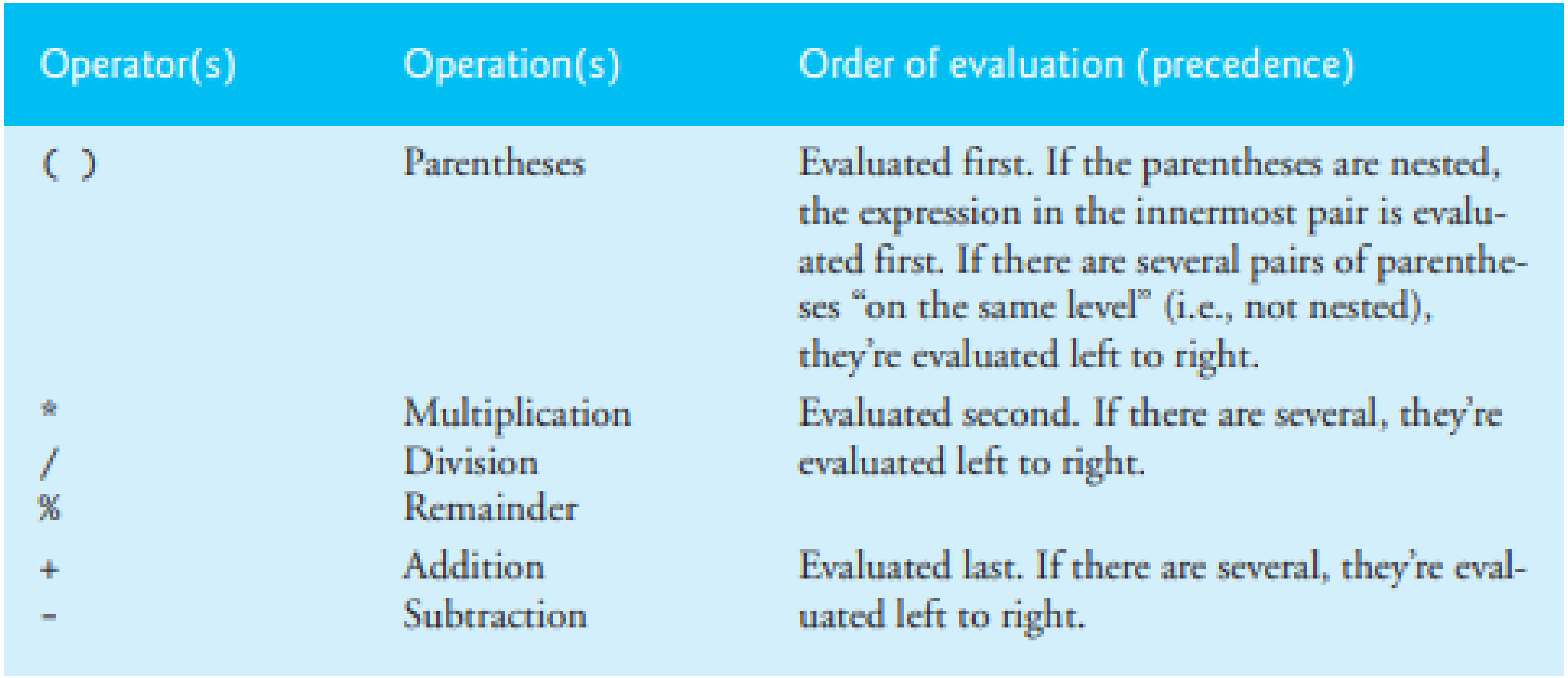
Operator precedence

Until now, we have used only one single operator in our programs

We know that quite often algebraic expression have multiple operators in a single expression

In C also, we can use multiple operators in a single expression using the concept of **Operator Precedence**

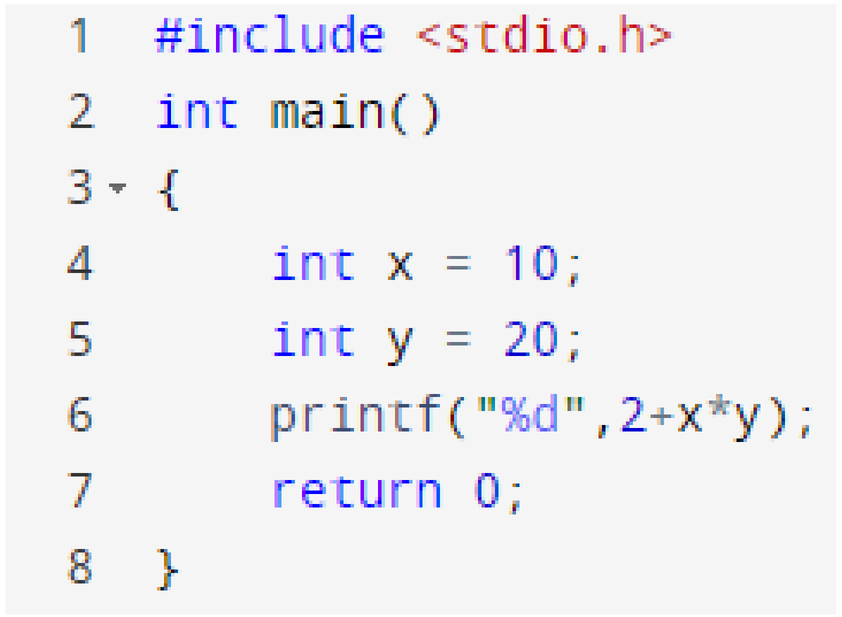
Operator precedence



Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Referred from : Deitel and Deitel – How to Program in C

Operator precedence 1



```
#include <stdio.h>
int main()
{
        int x = 10;
        int y = 20;
        printf("%d",2+x*y);
        return 0;
}
```

240

Output

202

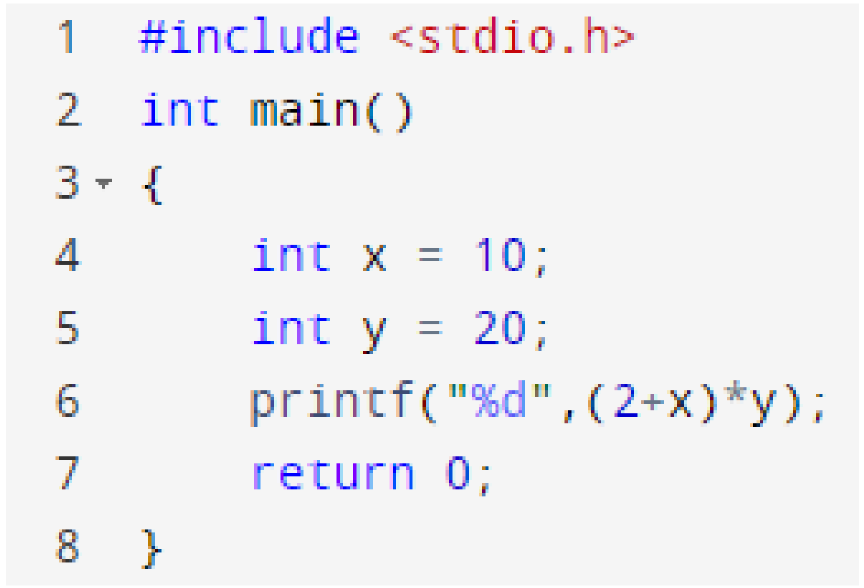
Output

Is the output 240 or 202 ?

Two operators (+ and *) are involved

What is the precedence of these operators ?

Operator precedence 2



```c
#include <stdio.h>
int main()
{
      int x = 10;
      int y = 20;
      printf("%d",(2+x)*y);
      return 0;
}
```

240

Output

202

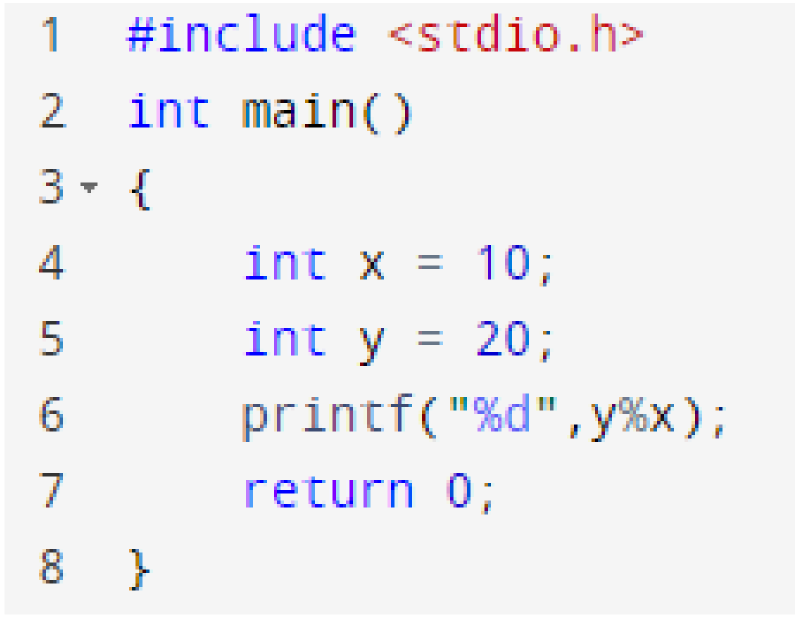
Output

Is the output 240 or 202 ?

Three operators +, * and () are involved

What is the precedence of these operators ?

Modulus operator 1



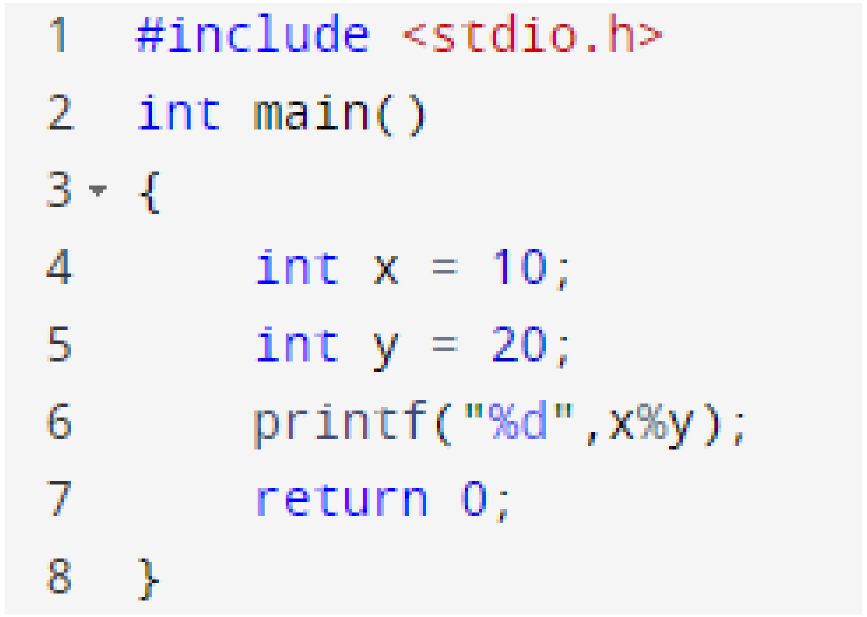
```c
#include <stdio.h>
int main()
{
        int x = 10;
        int y = 20;
        printf("%d",y%x);
        return 0;
}
```

0

Output

10

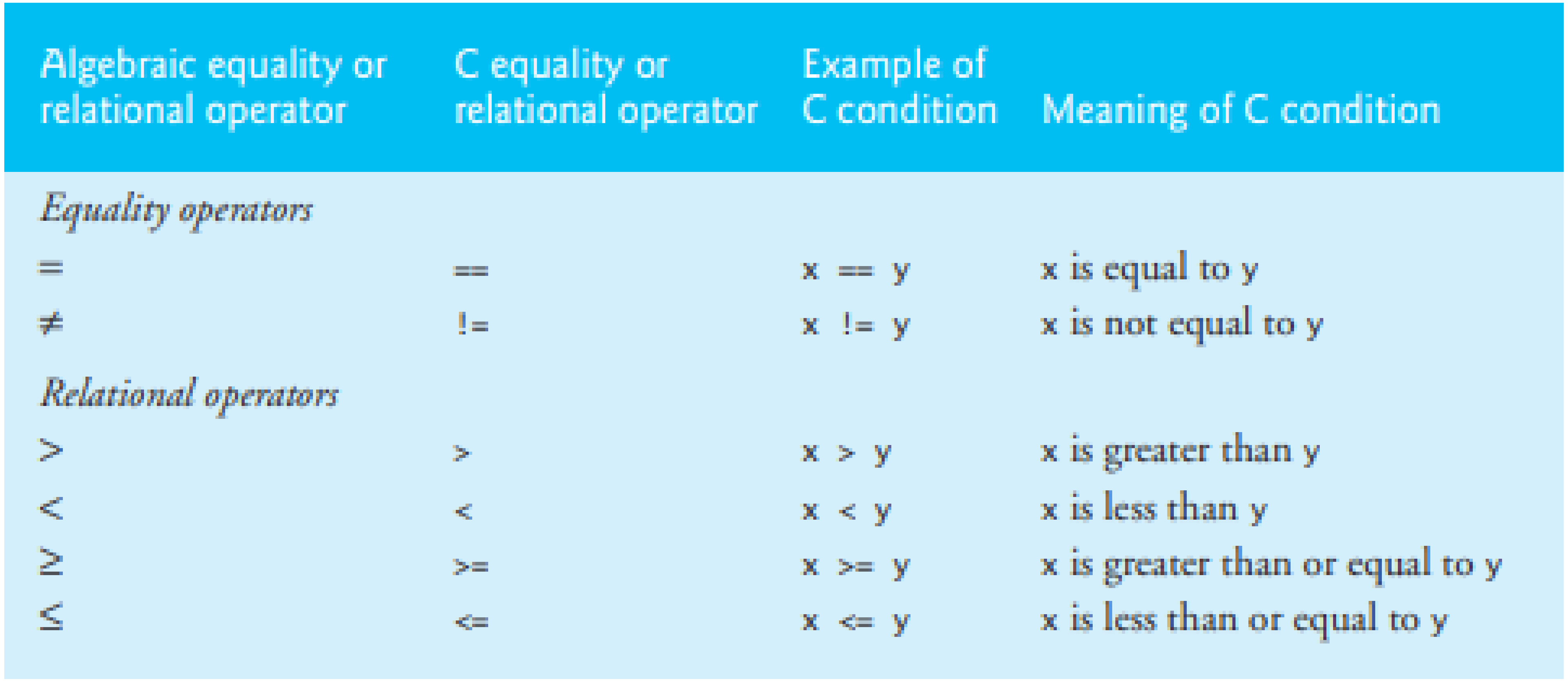
Output



```c
#include <stdio.h>
int main()
{
        int x = 10;
        int y = 20;
        printf("%d",x%y);
        return 0;
}
```

- Study the code
- Run the two programs
- **Note the output of % operator for the program on the right side**

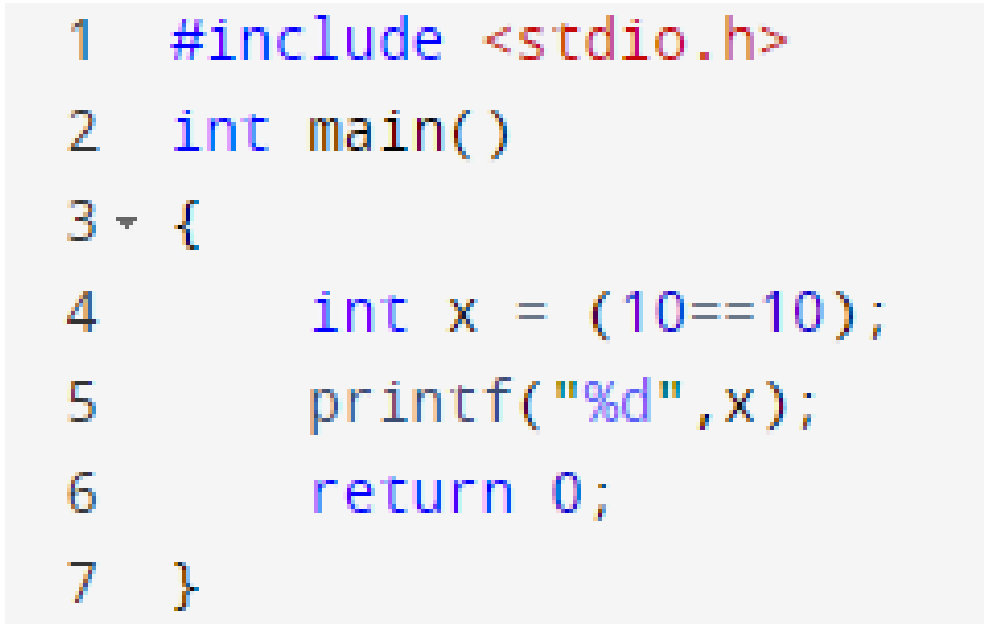
Operators and Decision-making statements



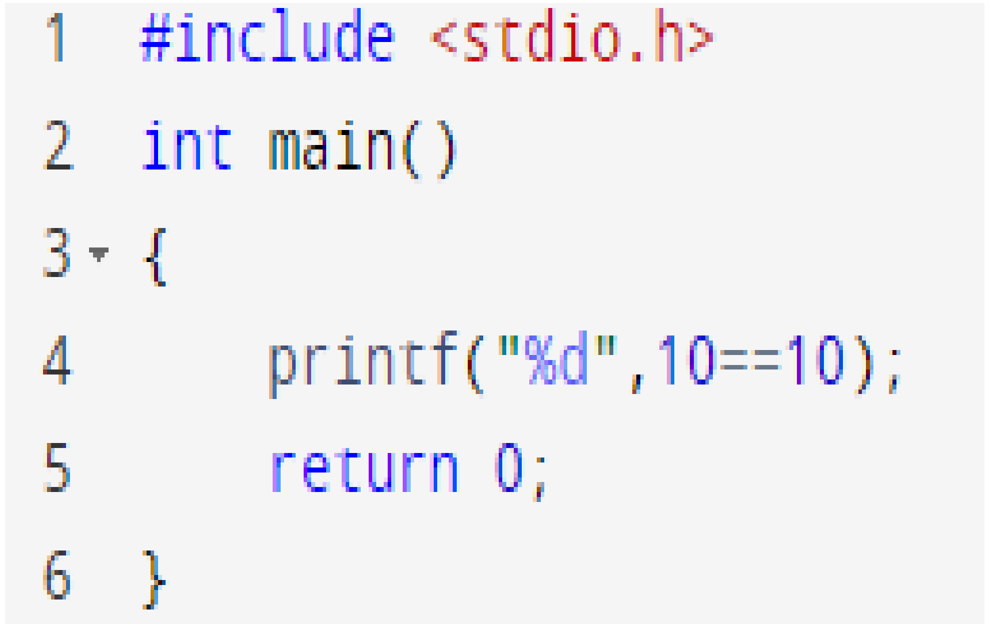
Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	`==`	`x == y`	x is equal to y
≠	`!=`	`x != y`	x is not equal to y
Relational operators			
>	`>`	`x > y`	x is greater than y
<	`<`	`x < y`	x is less than y
≥	`>=`	`x >= y`	x is greater than or equal to y
≤	`<=`	`x <= y`	x is less than or equal to y

Referred from : Deitel and Deitel – How to Program in C

Relational operators



```c
#include <stdio.h>
int main()
{
        int x = (10==10);
        printf("%d",x);
        return 0;
}
```

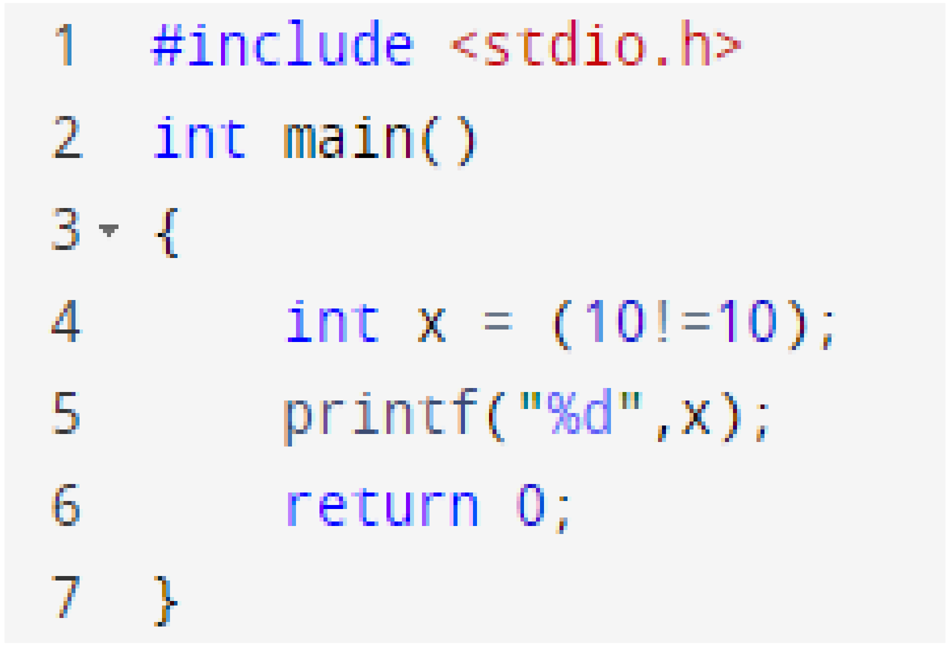


```c
#include <stdio.h>
int main()
{
        printf("%d",10==10);
        return 0;
}
```

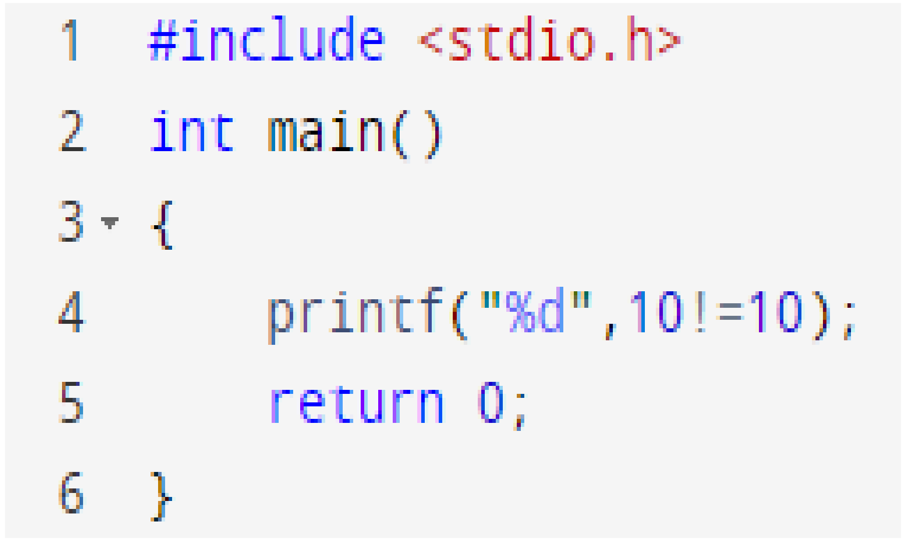
- Study the code
- Run the two programs
- Find the differences and similarity

What does the == operator return?

Relational operators



```c
#include <stdio.h>
int main()
{
      int x = (10!=10);
      printf("%d",x);
      return 0;
}
```

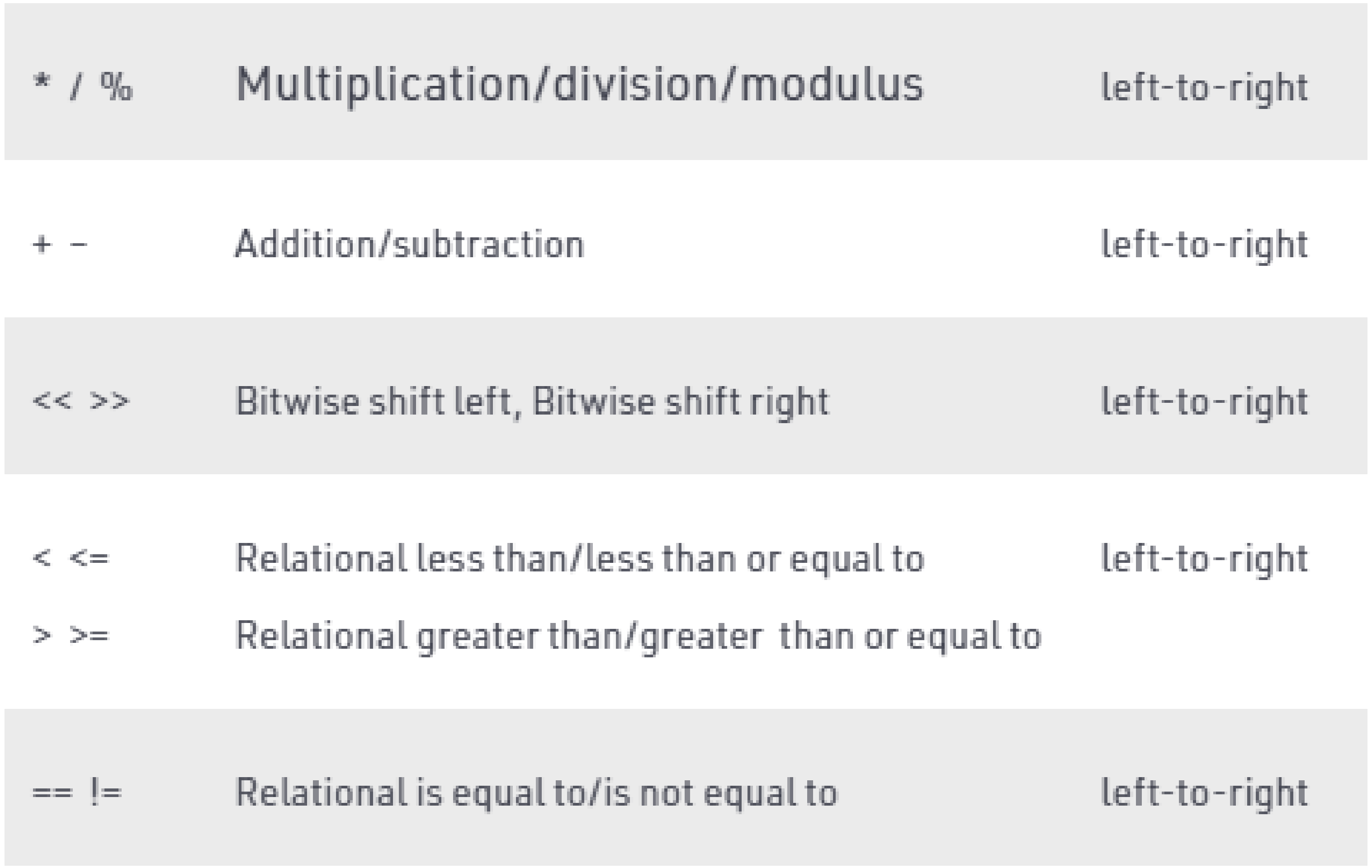


```c
#include <stdio.h>
int main()
{
      printf("%d",10!=10);
      return 0;
}
```

- Study the code
- Run the two programs
- Find the differences and similarity

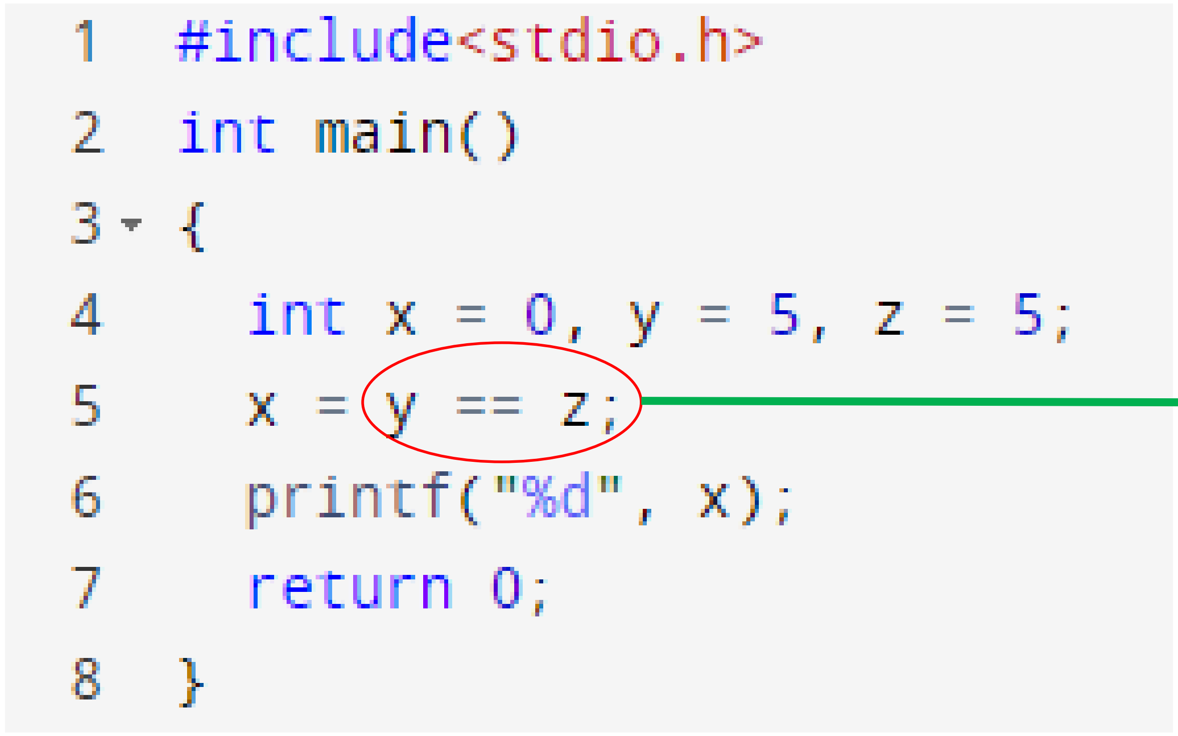
What does the != operator return?

Operator precedence – Arithmetic and Relational operators



* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right

Operator precedence – Arithmetic and Relational operators



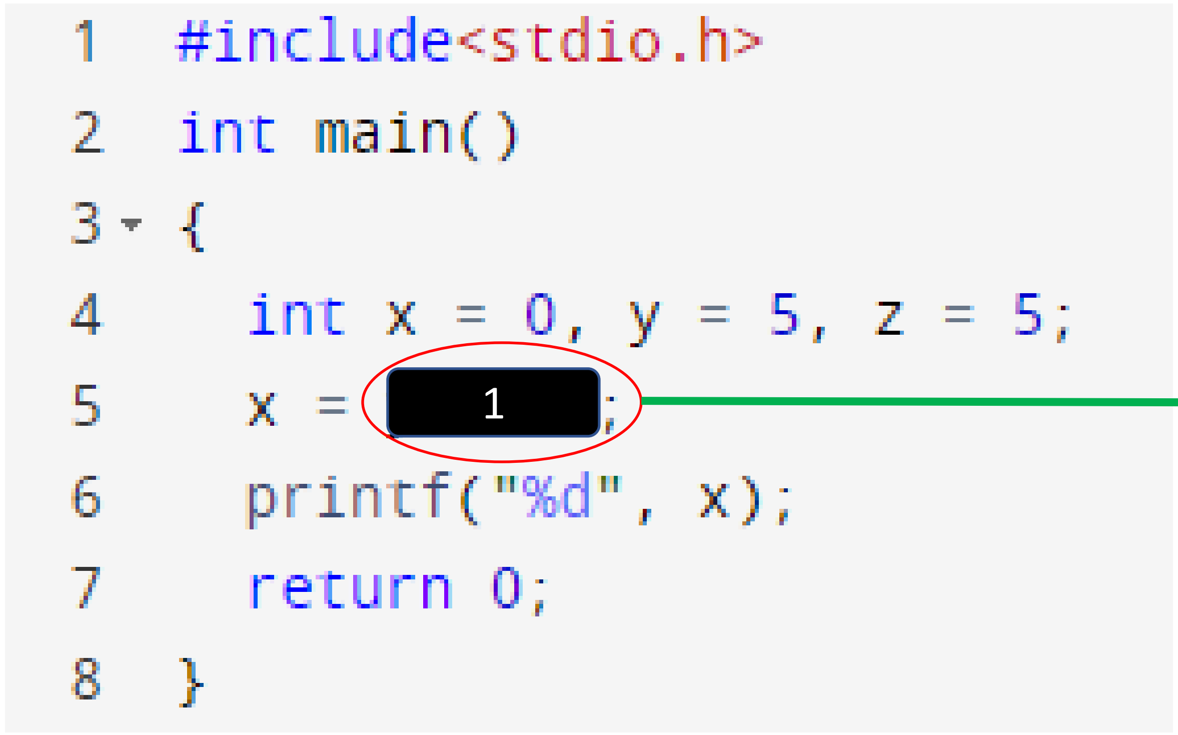
```c
#include<stdio.h>
int main()
{
    int x = 0, y = 5, z = 5;
    x = y == z;
    printf("%d", x);
    return 0;
}
```

Two operators = and ==

== has greater precedence than =

Is executed first

Operator precedence – Arithmetic and Relational operators



```c
#include<stdio.h>
int main()
{
    int x = 0, y = 5, z = 5;
    x = 1;
    printf("%d", x);
    return 0;
}
```

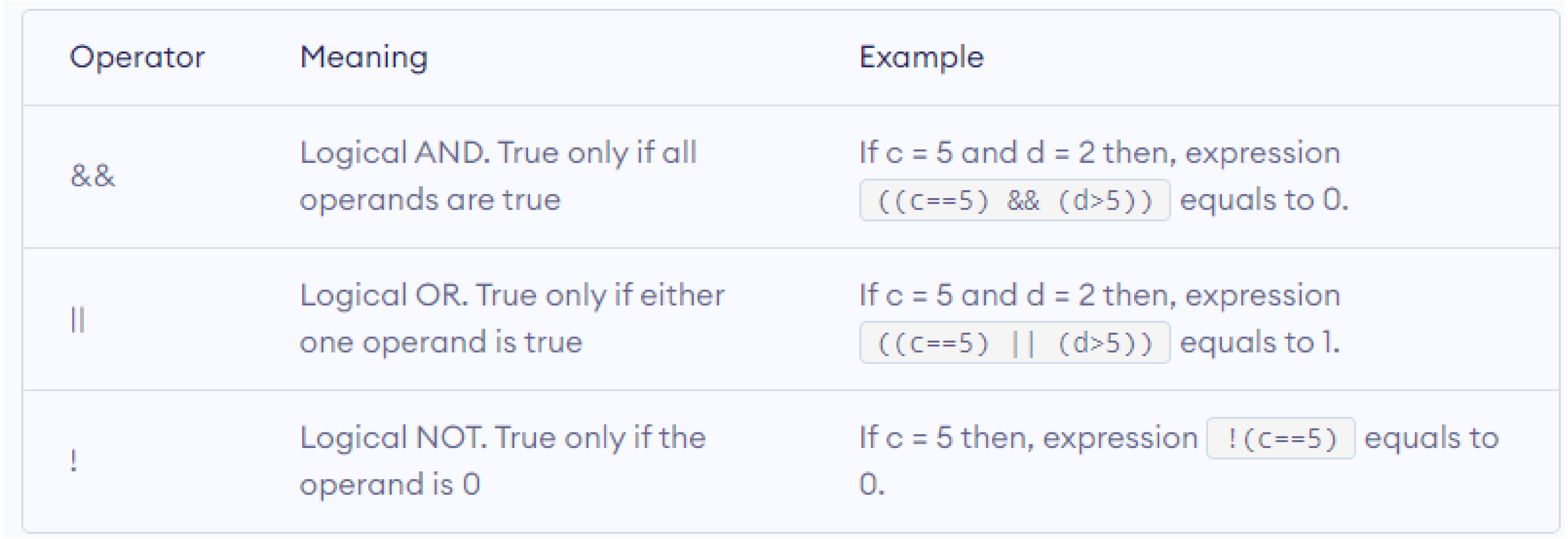
Two operators = and ==

== has greater precedence than =

y==z evaluates to 1

x is assigned the value 1

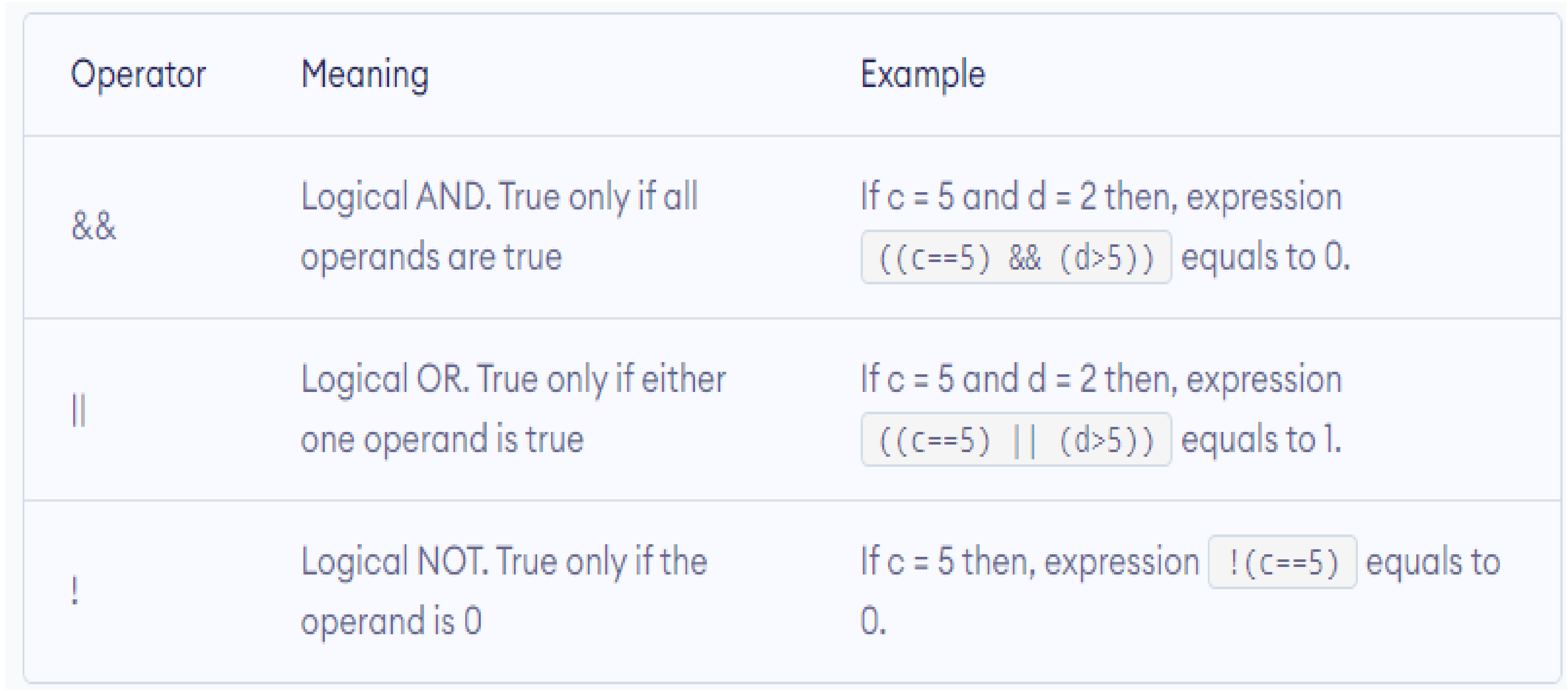
Logical operators



Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

Reference: https://www.programiz.com/c-programming/c-operators#logical

Logical operators



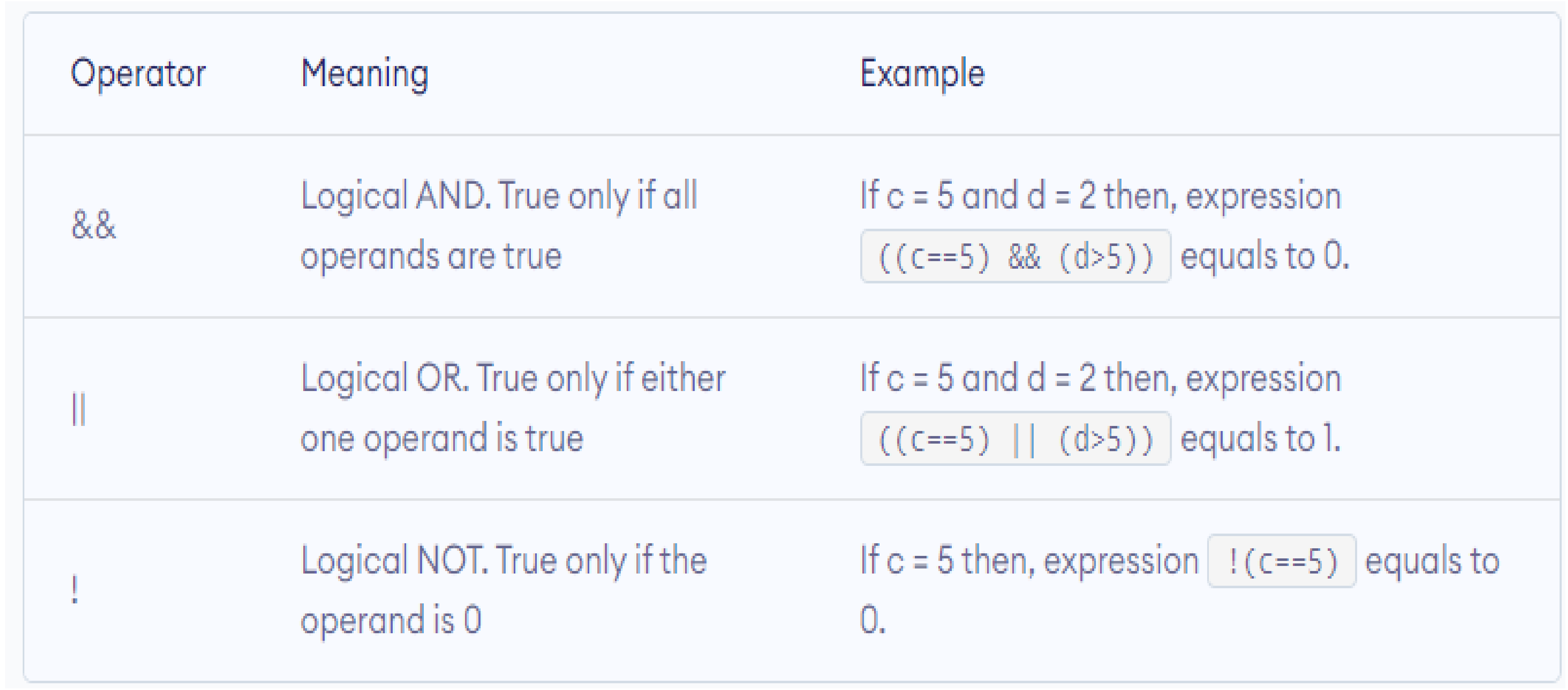
Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

Reference: https://www.programiz.com/c-programming/c-operators#logical

Arithmetic operators have higher precedence than Relational operators

Logical AND has higher precedence than Logical OR

Logical operators



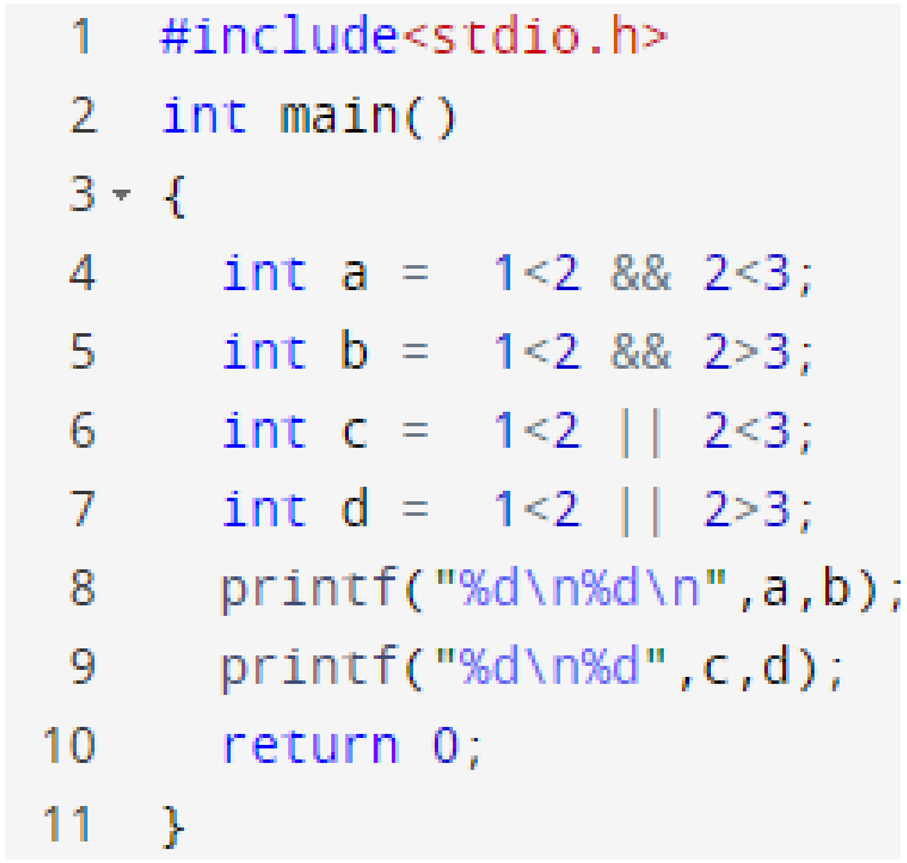
Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `!(c==5)` equals to 0.

Reference: https://www.programiz.com/c-programming/c-operators#logical

Arithmetic operators have higher precedence than Relational operators

Relational operators have higher precedence than Logical operators

Logical operators

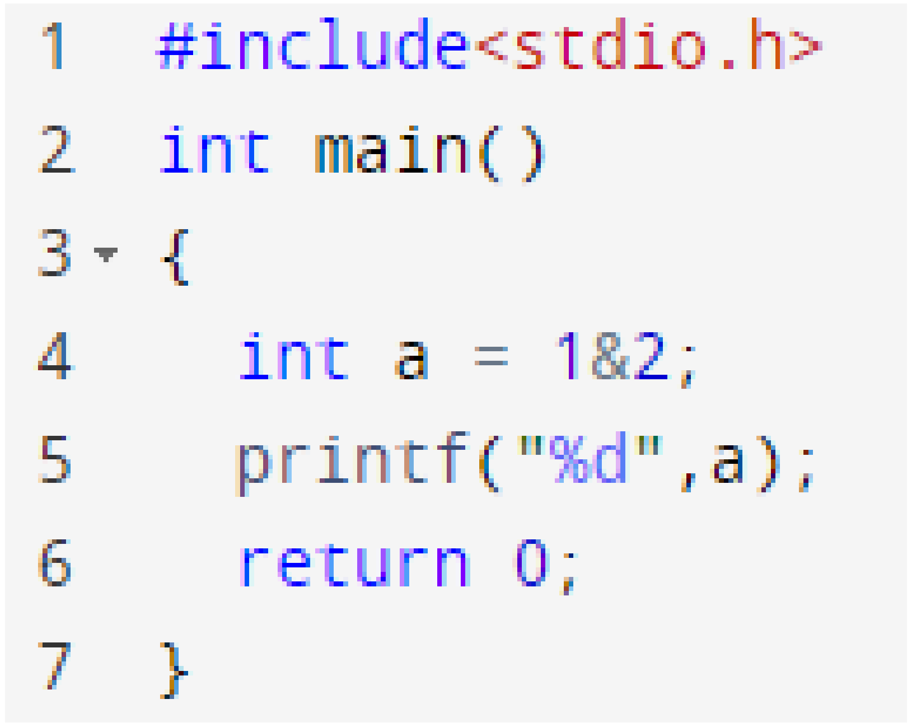


```
#include<stdio.h>
int main()
{
   int a =  1<2 && 2<3;
   int b =  1<2 && 2>3;
   int c =  1<2 || 2<3;
   int d =  1<2 || 2>3;
   printf("%d\n%d\n",a,b);
   printf("%d\n%d",c,d);
   return 0;
}
```

1<2 returns 1
2<3 returns 1
1 && 1 returns 1

1<2 returns 1
2>3 returns 0
1 && 0 returns 0

BITWISE operators



```
#include<stdio.h>
int main()
{
    int a = 1&2;
    printf("%d",a);
    return 0;
}
```

Bitwise AND

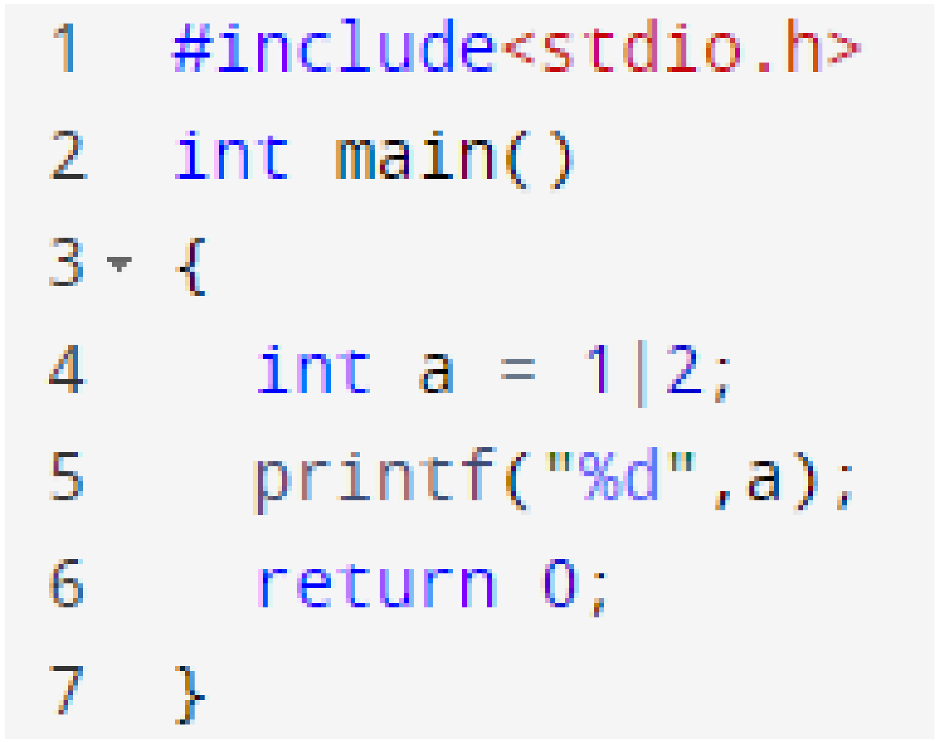
01
10

00

Different values give the output as 0

Same value will return the output 1

BITWISE operators



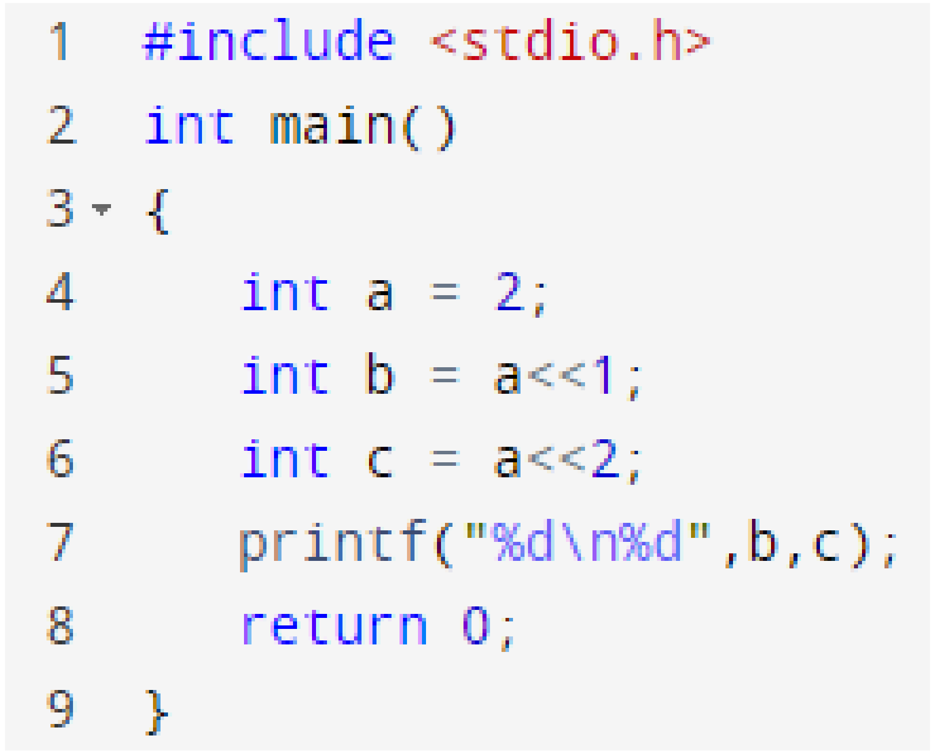
```c
#include<stdio.h>
int main()
{
    int a = 1|2;
    printf("%d",a);
    return 0;
}
```

Bitwise OR

```
 01
 10
--------
 11
```

0 OR 1 is 1
0 OR 0 is 0
1 OR 1 is 1
1 OR 0 is 1

BITWISE operators



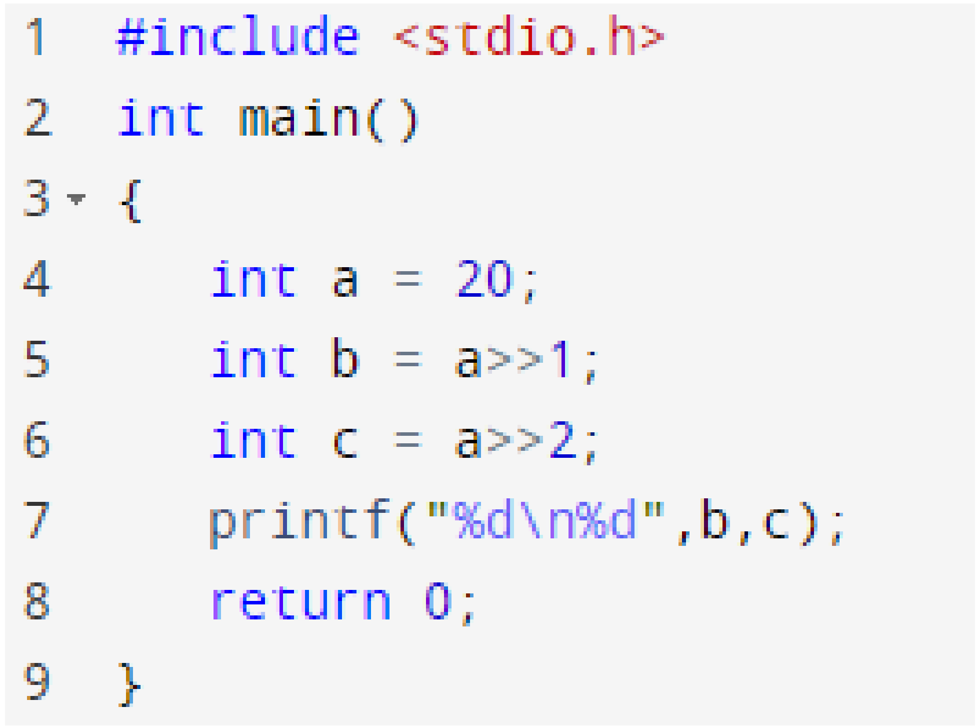
```c
#include <stdio.h>
int main()
{
    int a = 2;
    int b = a<<1;
    int c = a<<2;
    printf("%d\n%d",b,c);
    return 0;
}
```

Left Shift

Multiply a by 2^n

$2 \times 2^1 = 4$
$2 \times 2^2 = 8$

BITWISE operators



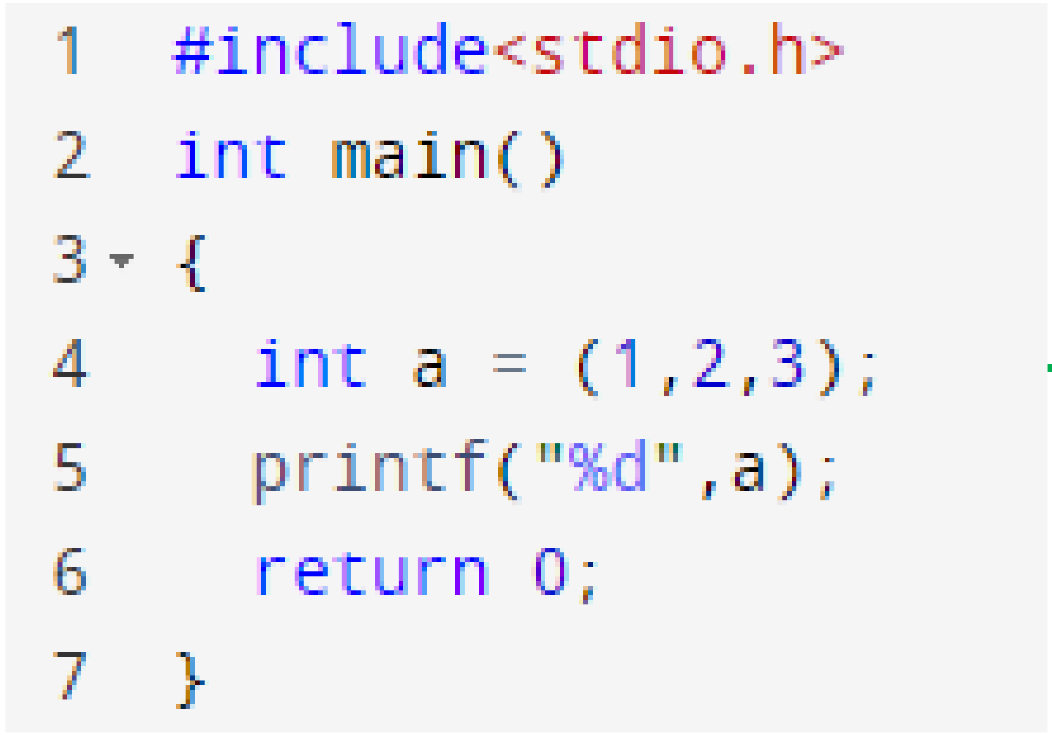
```
#include <stdio.h>
int main()
{
    int a = 20;
    int b = a>>1;
    int c = a>>2;
    printf("%d\n%d",b,c);
    return 0;
}
```

Right Shift

Divide a by 2^n

$20 / 2^1 = 10$
$20 / 2^2 = 5$

Comma operator



```c
#include<stdio.h>
int main()
{
    int a = (1,2,3);
    printf("%d",a);
    return 0;
}
```

Comma operator assigns the last value

Here the number 3 is assigned to the variable a

int a = (f1(), f2()) will evaluate f1 () but will return the value of f2() to a

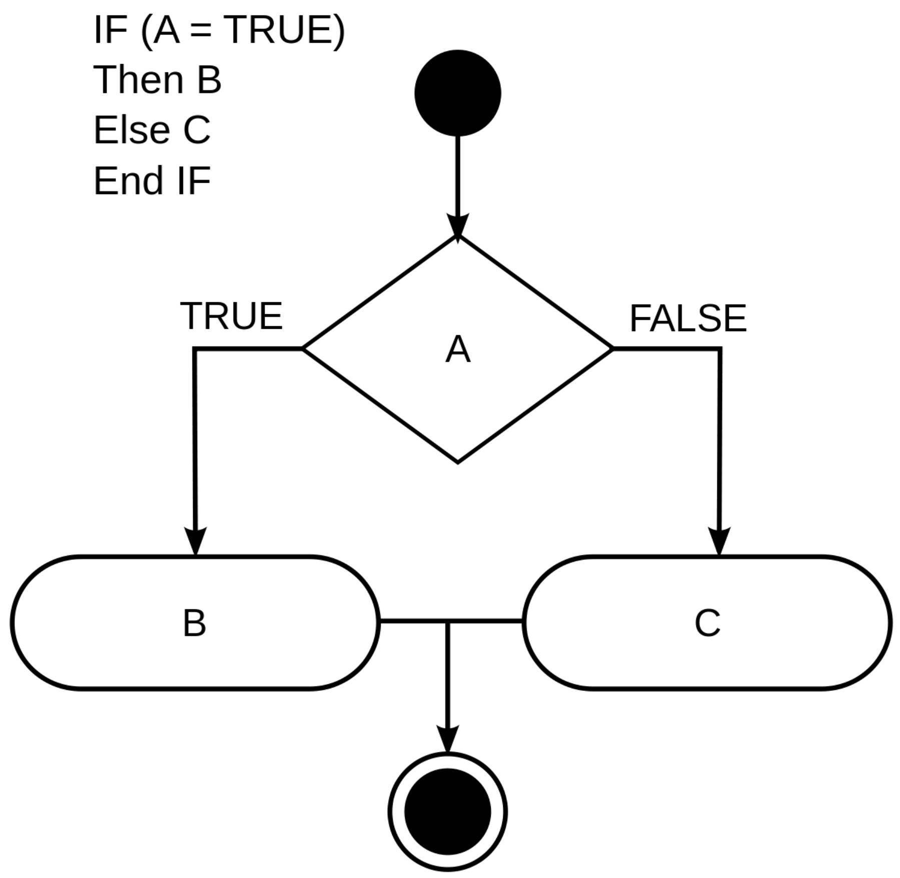
The comma operator has the lowest precedence of any C operator

Operators and Decision-making statements

On many occasions we need the computer to make decisions

Decisions are based on the Truth or Falsity of a statement called a condition

Operators and Decision-making statements

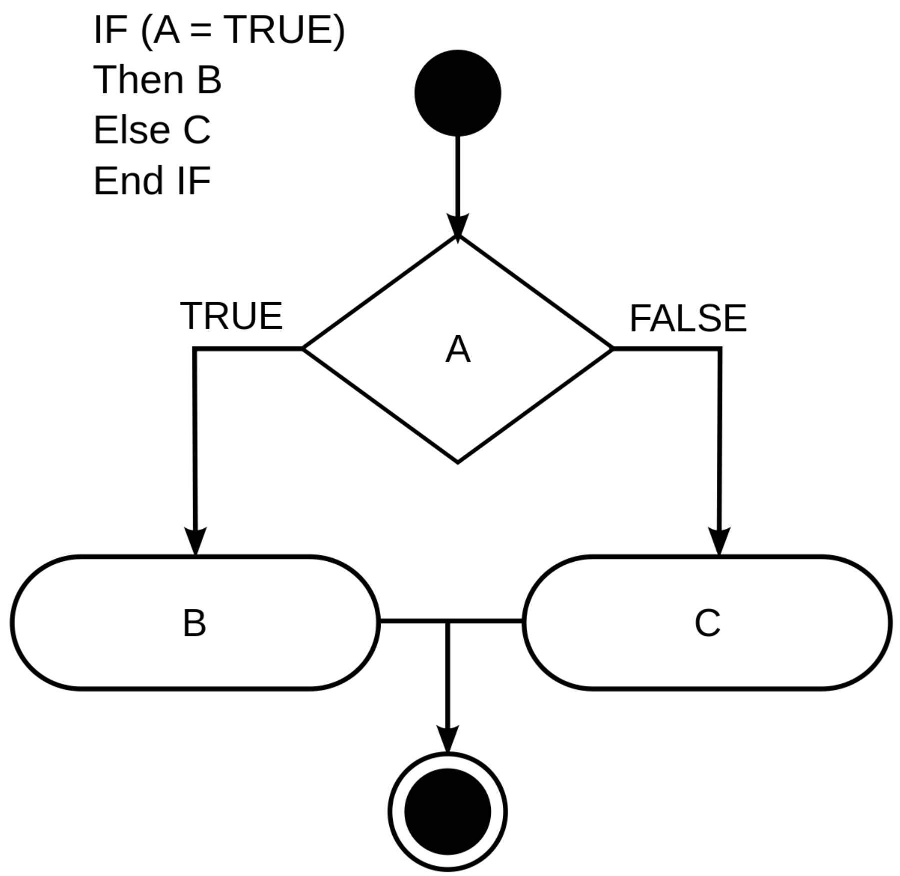


From: wikipedia.org

IF statement is used to make decisions based on a condition

Conditions are formed by using Operators

Operators and Decision-making statements

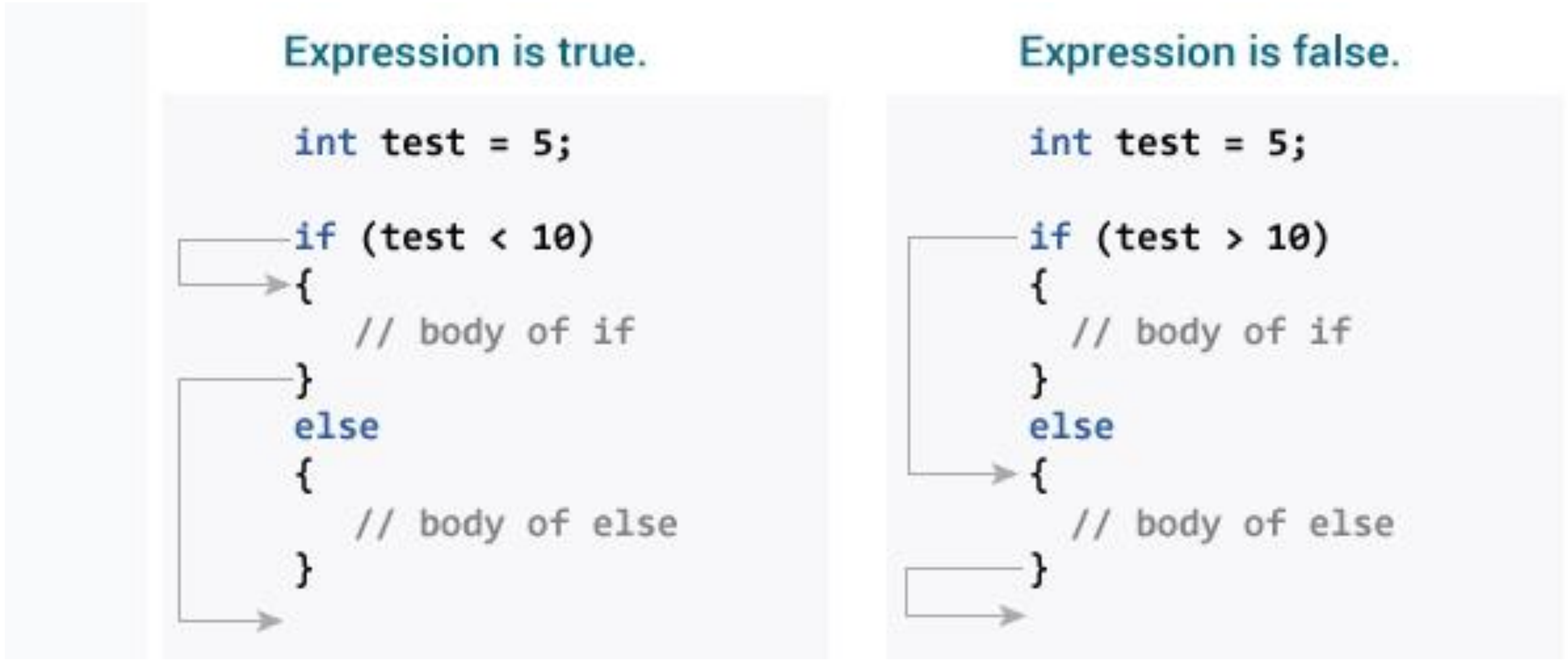


From: wikipedia.org

IF statement is used to make decisions based on a condition

Equality or Relational operators

If Else

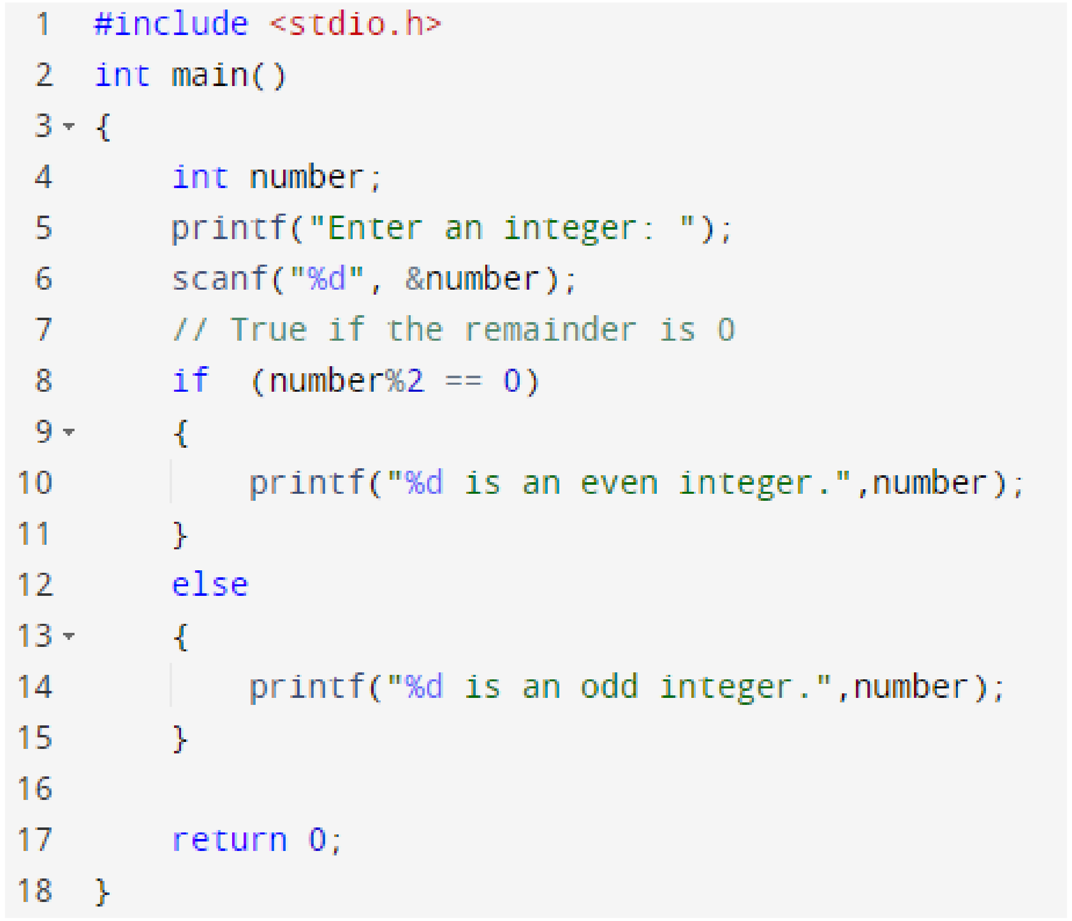


```
int test = 5;

if (test < 10)
{
   // body of if
}
else
{
   // body of else
}
```

```
int test = 5;

if (test > 10)
{
   // body of if
}
else
{
   // body of else
}
```

Reference: programiz.com

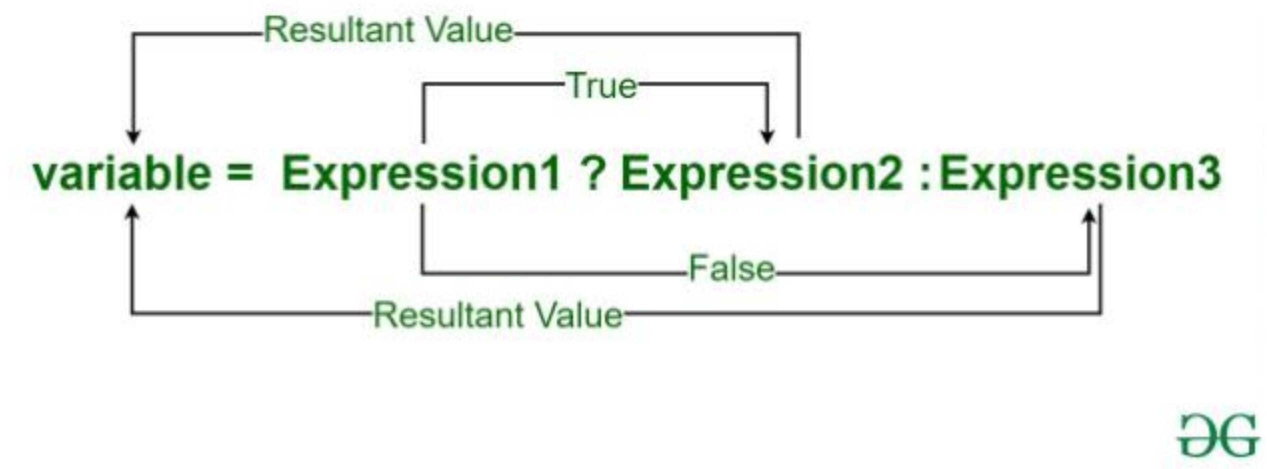
If Else



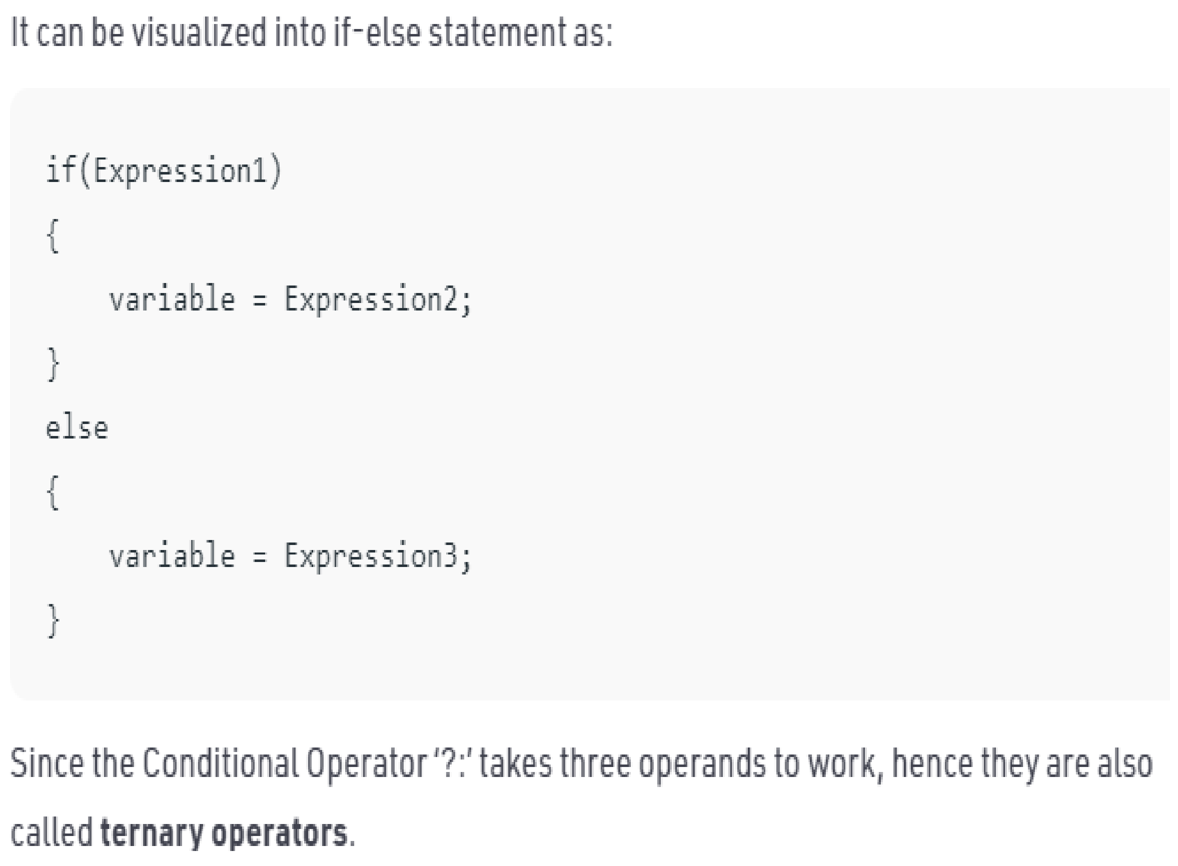
```c
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
    // True if the remainder is 0
    if  (number%2 == 0)
    {
        printf("%d is an even integer.",number);
    }
    else
    {
        printf("%d is an odd integer.",number);
    }

    return 0;
}
```

Reference: programiz.com

Conditional operator



Reference: geekforgeeks.org



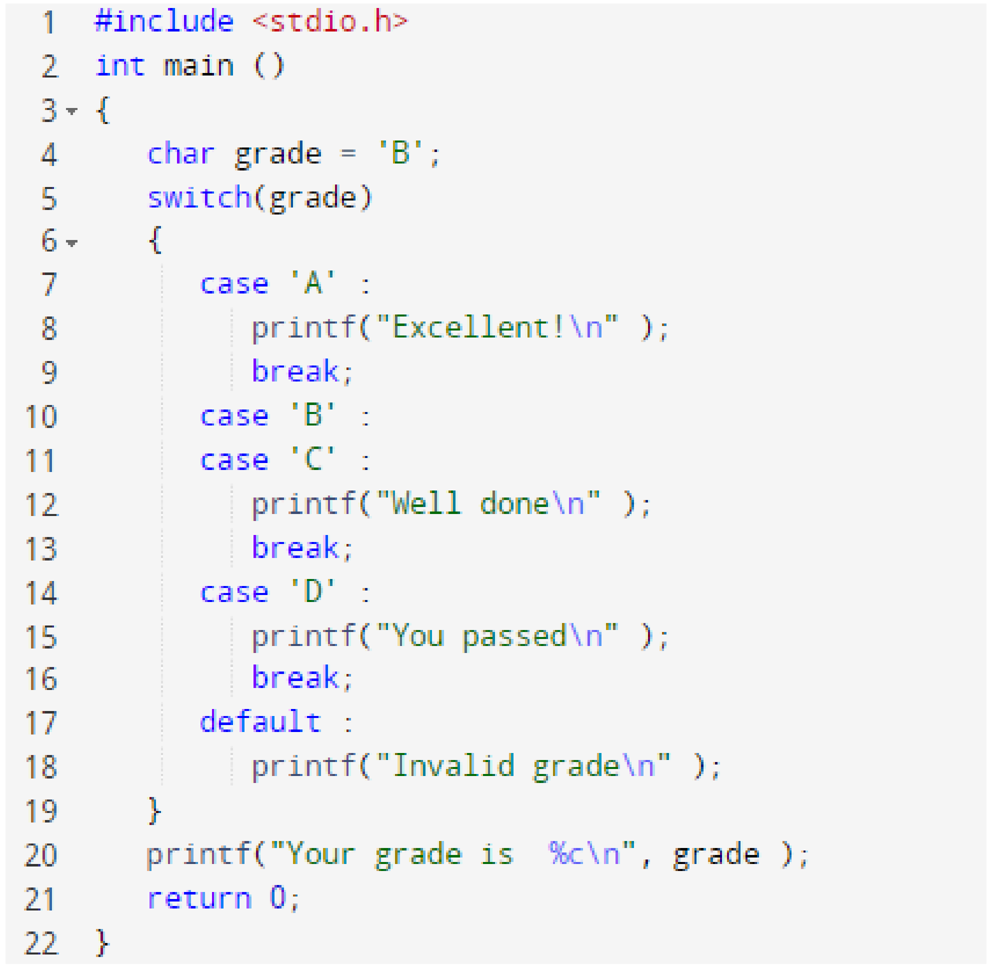
It can be visualized into if-else statement as:

```
if(Expression1)
{
    variable = Expression2;
}
else
{
    variable = Expression3;
}
```

Since the Conditional Operator '?:' takes three operands to work, hence they are also called **ternary operators**.

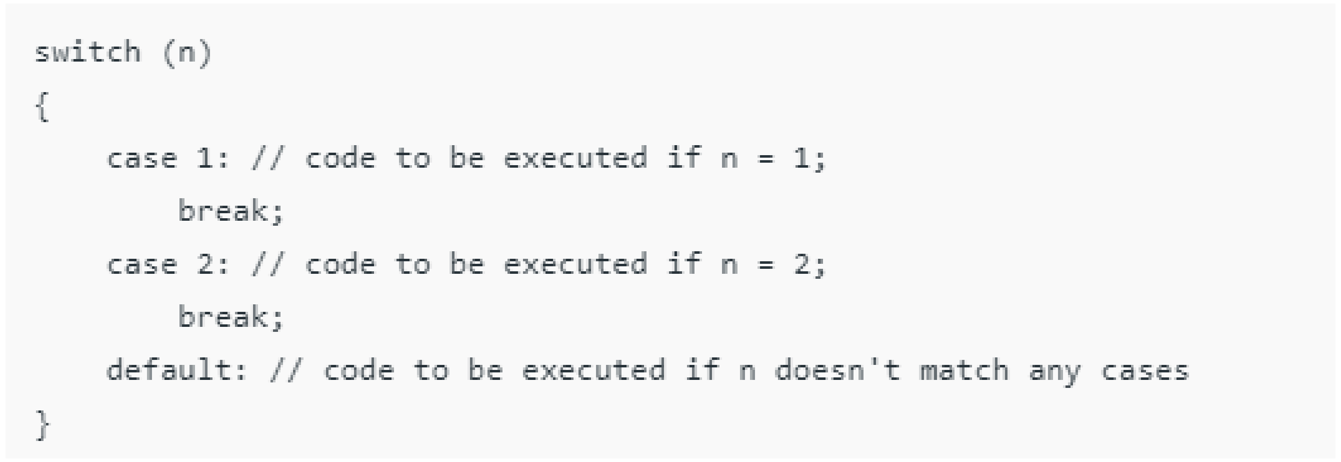
Reference: geekforgeeks.org

Switch



```c
#include <stdio.h>
int main ()
{
    char grade = 'B';
    switch(grade)
    {
        case 'A' :
            printf("Excellent!\n" );
            break;
        case 'B' :
        case 'C' :
            printf("Well done\n" );
            break;
        case 'D' :
            printf("You passed\n" );
            break;
        default :
            printf("Invalid grade\n" );
    }
    printf("Your grade is  %c\n", grade );
    return 0;
}
```

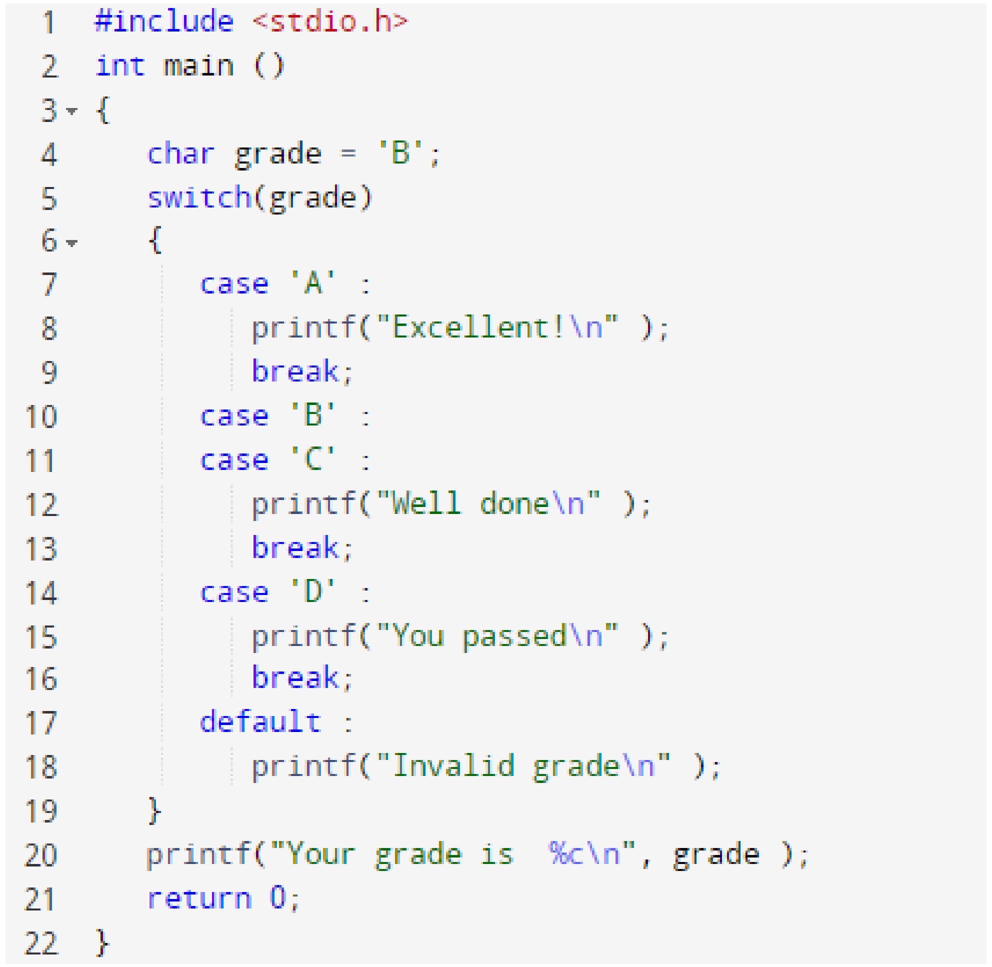
Reference: programiz.com



```
switch (n)
{
    case 1: // code to be executed if n = 1;
        break;
    case 2: // code to be executed if n = 2;
        break;
    default: // code to be executed if n doesn't match any cases
}
```

Reference: geekforgeeks.org

Switch



```
#include <stdio.h>
int main ()
{
    char grade = 'B';
    switch(grade)
    {
        case 'A' :
            printf("Excellent!\n" );
            break;
        case 'B' :
        case 'C' :
            printf("Well done\n" );
            break;
        case 'D' :
            printf("You passed\n" );
            break;
        default :
            printf("Invalid grade\n" );
    }
    printf("Your grade is  %c\n", grade );
    return 0;
}
```

Reference: programiz.com

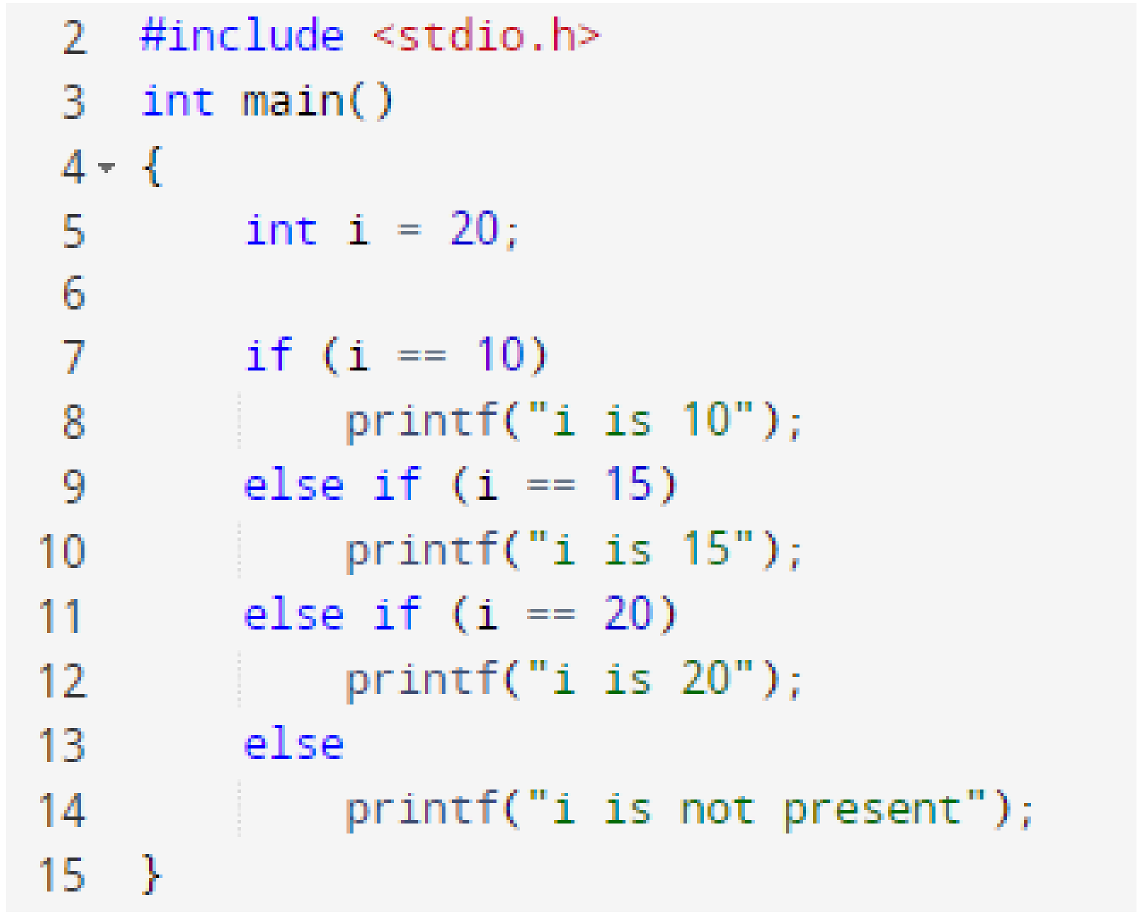
1.Duplicate case values are not allowed.

2.The default statement is optional.

3.The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.

4.The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

If Else If ladder



```c
#include <stdio.h>
int main()
{
    int i = 20;

    if (i == 10)
        printf("i is 10");
    else if (i == 15)
        printf("i is 15");
    else if (i == 20)
        printf("i is 20");
    else
        printf("i is not present");
}
```

The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed