

# Programming in c++

## Lecture\_2

**Dr. Sara Mohamed**

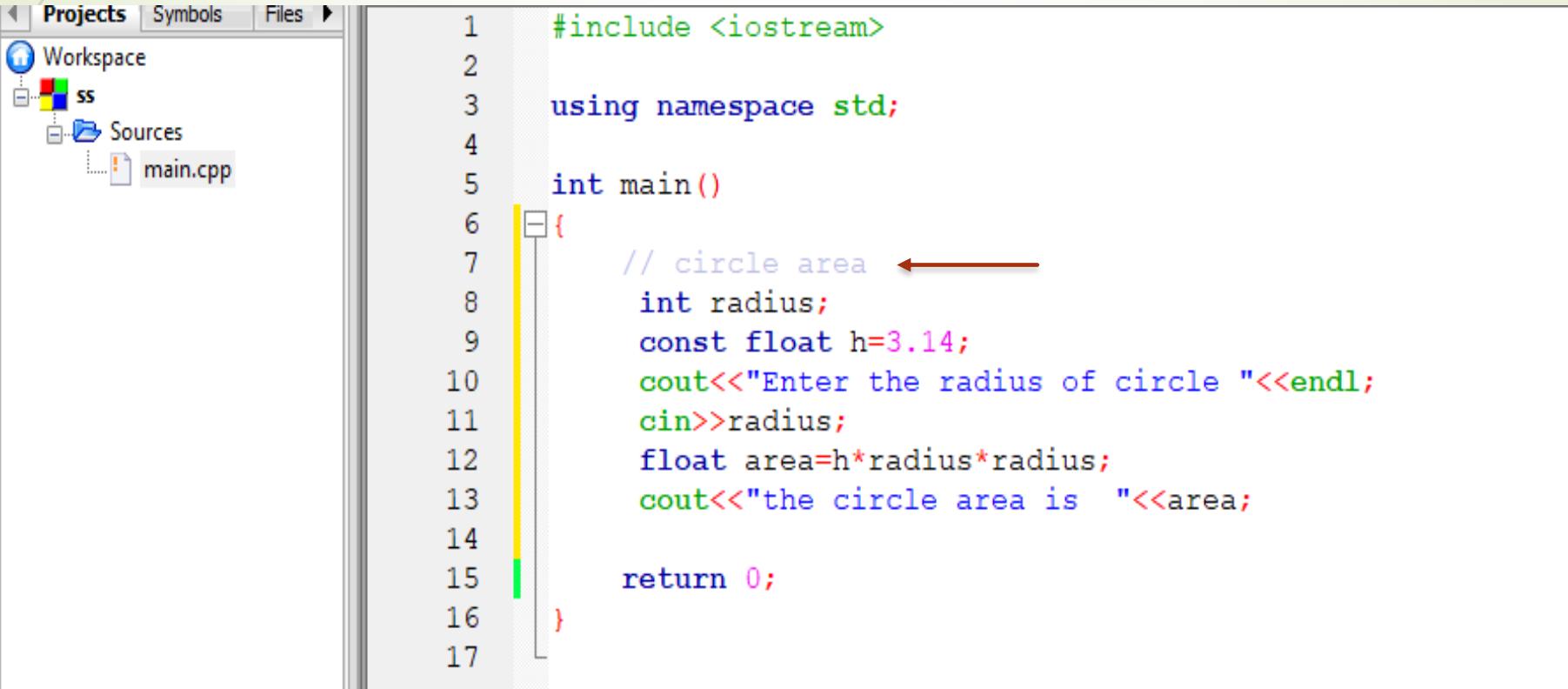


# C++ Comments

- ▶ **Comments** can be used to explain C++ code
- ▶ to make it more readable. It can also be used to prevent execution when testing alternative code.
- ▶ Comments can be **singled-lined or multi-lined**.

# Single-line comments

- ▶ **Single-line** comments start with two forward slashes (`//`).
- ▶ Any text between `//` and the end of the line is ignored by the compiler (will not be executed)



The screenshot shows a code editor interface with a 'Projects' tab selected. Under the 'Workspace' section, there is a project named 'ss' containing a source file 'main.cpp'. The code editor displays the following C++ code:

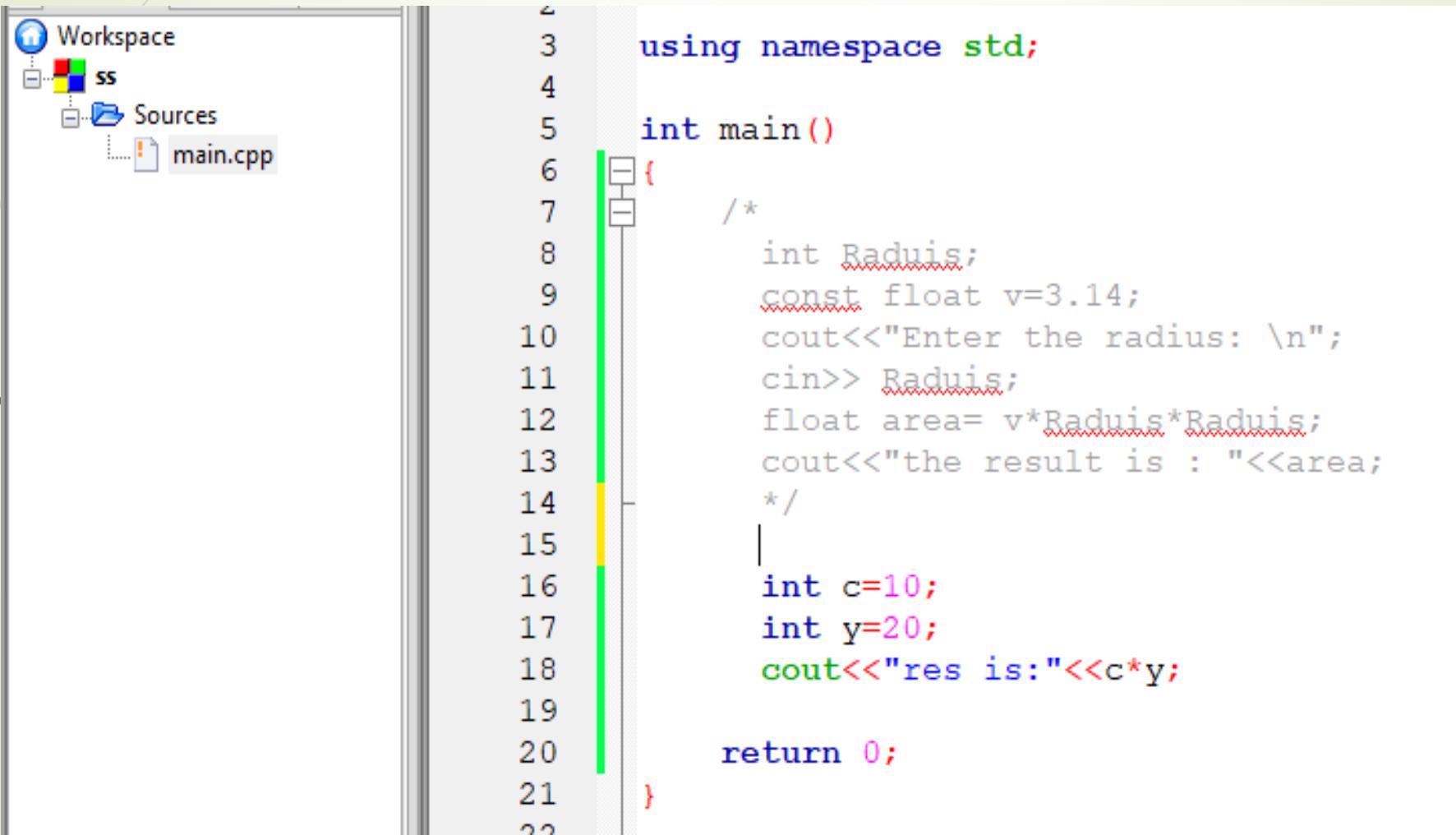
```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     // circle area ←
8     int radius;
9     const float h=3.14;
10    cout<<"Enter the radius of circle "<<endl;
11    cin>>radius;
12    float area=h*radius*radius;
13    cout<<"the circle area is  "<<area;
14
15    return 0;
16
17 }
```

A red arrow points to the line starting with `// circle area`, indicating that the text from the double slash to the end of the line is a single-line comment.

# C++ Multi-line Comments

Multi-line comments start with `/*` and ends with `*/`.

Any text between `/*` and `*/` will be ignored by the compiler.



```
3  using namespace std;
4
5  int main()
6  {
7      /*
8          int RADIUS;
9          const float pi=3.14;
10         cout<<"Enter the radius: \n";
11         cin>> RADIUS;
12         float area= pi*RADIUS*RADIUS;
13         cout<<"the result is : "<<area;
14     */
15
16     int c=10;
17     int y=20;
18     cout<<"res is:"<<c*y;
19
20     return 0;
21 }
```



# C++ Operators

- ▶ Operators are used to perform operations on variables and values.
- ▶ C++ divides the operators into the following groups:
  - ▶ **Arithmetic operators**
  - ▶ **Assignment operators**
  - ▶ **Comparison operators**
  - ▶ **Logical operators**

Name	Symbol	Arity	Usage
Add	+	binary	$x + y$
Subtract	-	binary	$x - y$
Multiply	*	binary	$x * y$
Divide	/	binary	$x / y$
Modulus	%	binary	$x \% y$
Minus	-	unary	$-x$

# Operators

- ▶ Special built-in symbols that have functionality, and work on operands
- ▶ **operand** - an input to an operator
- ▶ **Arity** - how many operands an operator takes
  - ▶ *unary operator* - has one operand
  - ▶ *binary operator* - has two operands
  - ▶ *ternary operator* - has three operands
- ▶ Examples:

---

```
int x, y = 5, z;
z = 10; // assignment operator (binary)
x = y + z; // addition (binary operator)
x = -y; // -y is a unary operation (negation)
x++; // unary (increment)
```

---

# Unary operators perform an operation using a single variable or value.

- A **unary operator** in C++ is an operator that works on **only one operand**.

## ◆ Common Unary Operators in C++

Operator	Name	Example	Description
<code>++</code>	Increment	<code>++x</code> or <code>x++</code>	Increases the value of <code>x</code> by 1
<code>--</code>	Decrement	<code>--x</code> or <code>x--</code>	Decreases the value of <code>x</code> by 1
<code>-</code>	Unary minus	<code>-x</code>	Changes the sign of <code>x</code>
<code>+</code>	Unary plus	<code>+x</code>	Indicates a positive value (rarely used)
<code>!</code>	Logical NOT	<code>!x</code>	Reverses a boolean value ( <code>true</code> → <code>false</code> )



The operator = works between these **two operands**, so it's **binary**.

A **binary operator** is an operator that operates on **two operands**.

For example:

cpp

```
x = y;
```

Here:

- `x` → **left operand** (the variable being assigned to)
- `y` → **right operand** (the value being assigned)

The operator `=` works between these **two operands**, so it's **binary**.

# Arithmetic Operators

- Arithmetic operators are used to perform common mathematical operations.

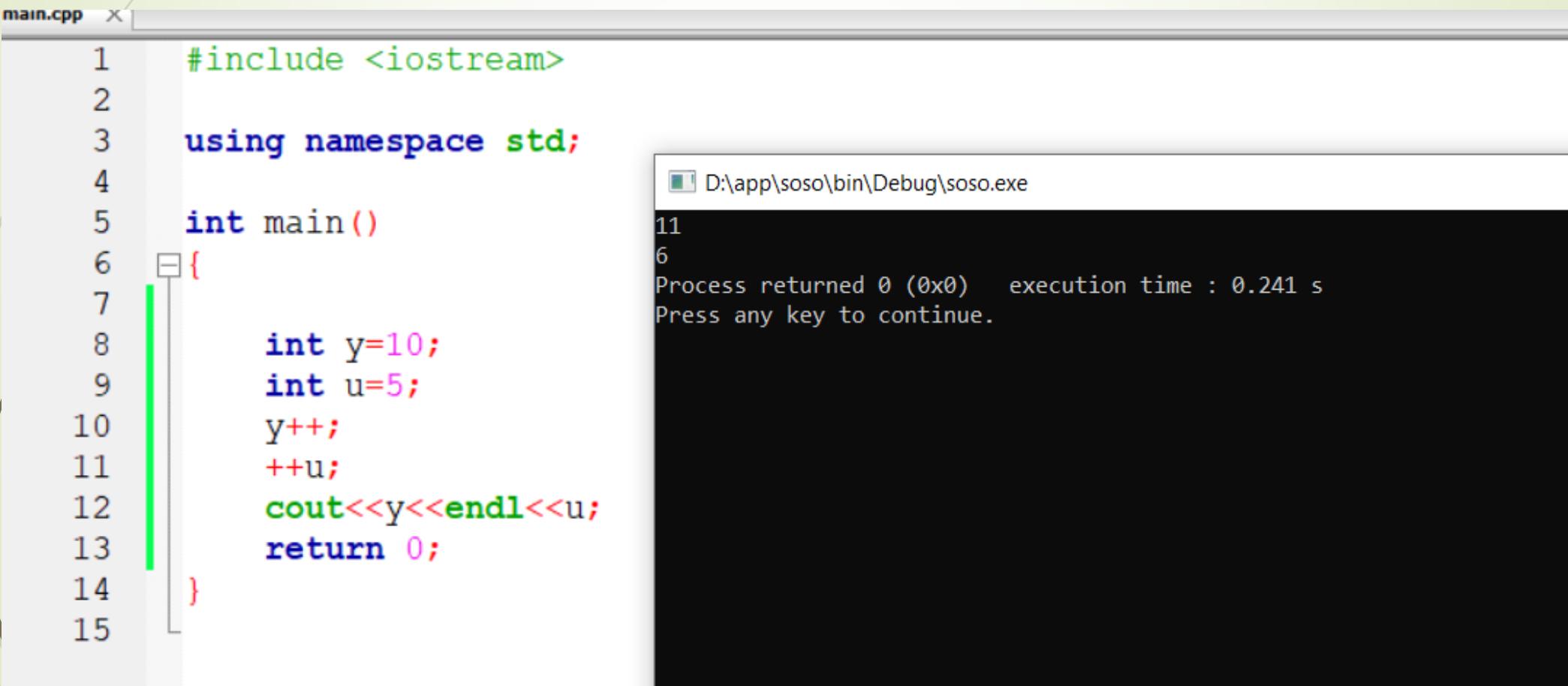
## Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value from another	$x / y$
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$++x$
--	Decrement	Decreases the value of a variable by 1	$--x$

# Increment and decrement

- ▶ **Increment** : increase the variable value by 1



```
main.cpp X
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int y=10;
8     int u=5;
9     y++;
10    ++u;
11    cout<<y<<endl<<u;
12
13    return 0;
14
15 }
```

D:\app\soso\bin\Debug\soso.exe

```
11
6
Process returned 0 (0x0) execution time : 0.241 s
Press any key to continue.
```

# Decrement

- **Decrement** : decrease the variable value by 1

The screenshot shows a C++ development environment with two panes. The left pane displays the source code of a program named 'soso'. The right pane shows the terminal output of the program's execution.

**Source Code (Left Pane):**

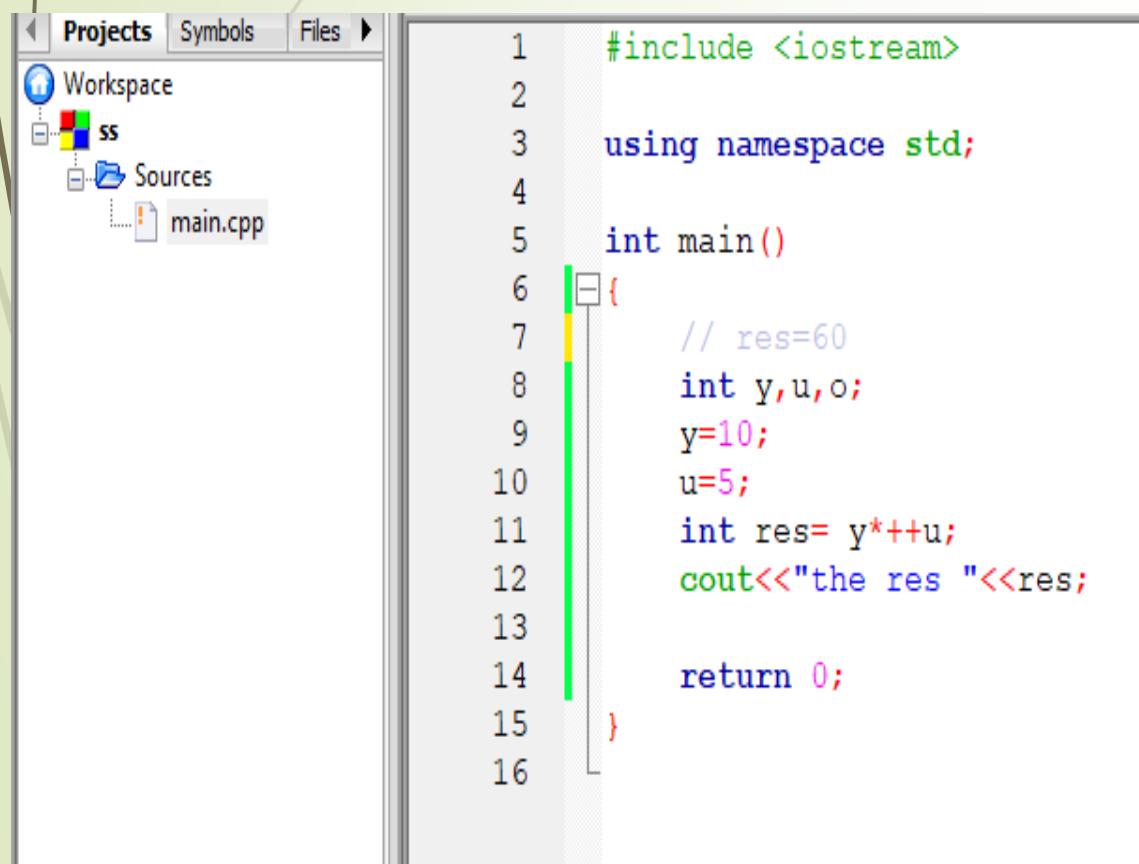
```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7
8     int y=10;
9     int u=5;
10
11    y--;
12    --u;
13
14    cout<<y<<endl<<u;
15
16    return 0;
17 }
```

**Terminal Output (Right Pane):**

```
D:\app\soso\bin\Debug\soso.exe
9
4
Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

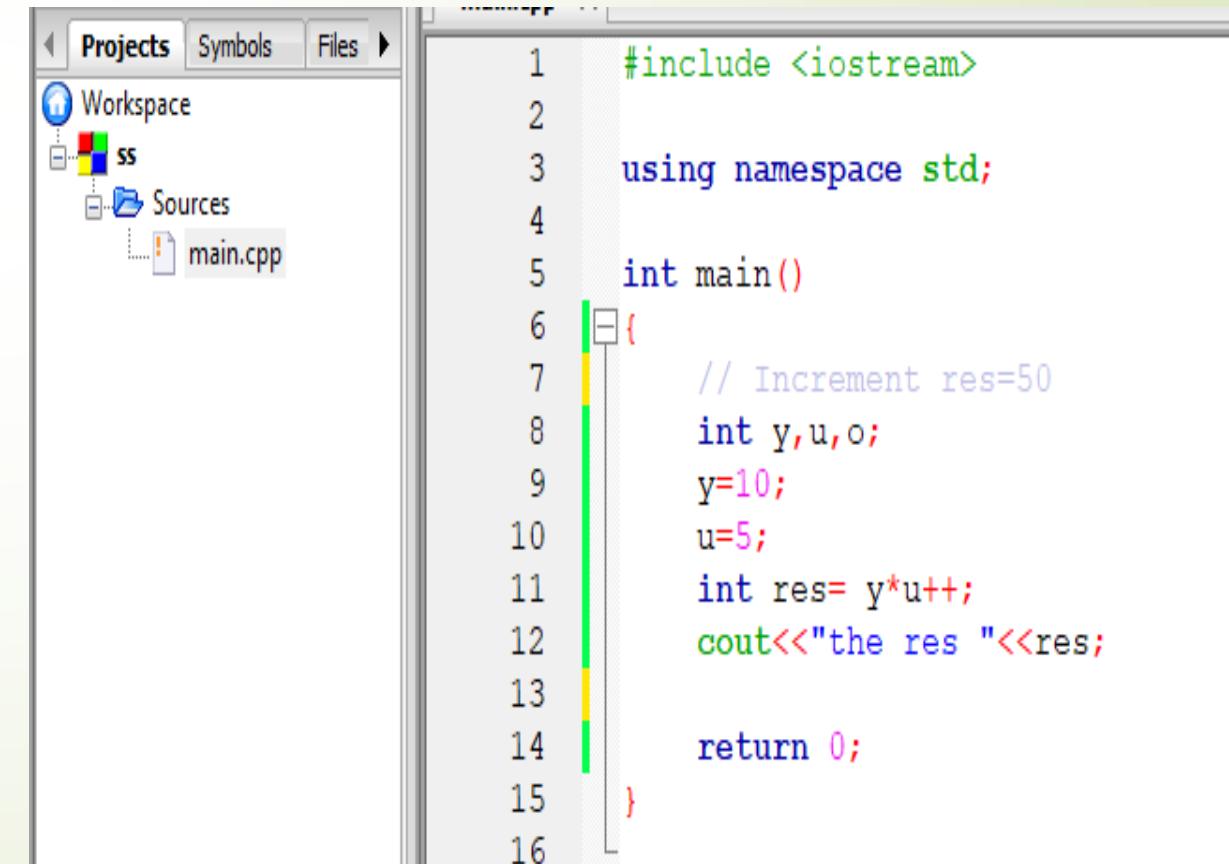
# Difference between $++x$ and $x++$

both  $++x$  and  $x++$  are used to increment variable  $x$  by 1.. the prime difference is that:  
 $++x$  i.e. **pre-increment** operator uses the principle ‘change-then-use’ while,  $x++$  i.e.  
**post-increment** operator uses the principle ‘use-then-change’.



Projects Symbols Files  
Workspace ss Sources main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     // res=60
8     int y,u,o;
9     y=10;
10    u=5;
11    int res= y++u;
12    cout<<"the res "<<res;
13
14    return 0;
15 }
16
```



Projects Symbols Files  
Workspace ss Sources main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     // Increment res=50
8     int y,u,o;
9     y=10;
10    u=5;
11    int res= y*u++;
12    cout<<"the res "<<res;
13
14    return 0;
15 }
16
```

# Difference between --x and x--

A screenshot of a code editor showing a C++ file named main.cpp. The code contains a bug where the variable 'res' is calculated using pre-increment (y++) instead of post-increment (y--). The code is as follows:

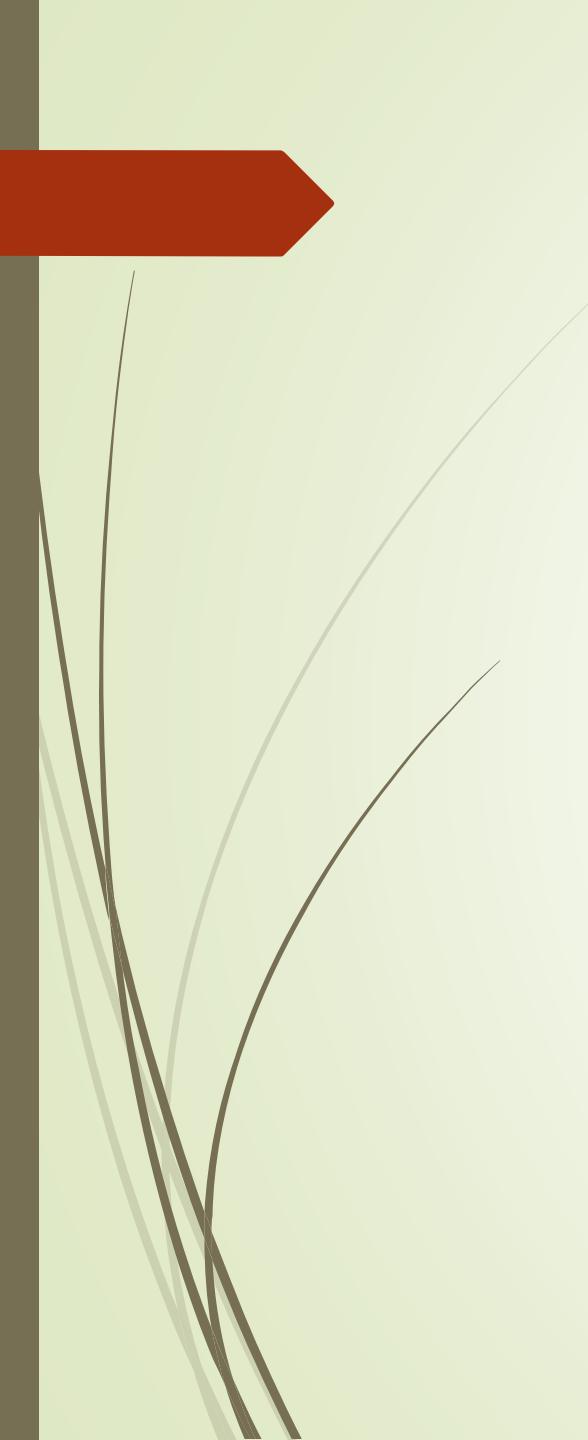
```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     // res=50
8     int y,u,o;
9     y=10;
10    u=5;
11    int res= y*u--;
12    cout<<"the res "<<res;
13
14    return 0;
15 }
16
```

The output of this code would be 50, but it should be 40.

A screenshot of a code editor showing the same C++ file main.cpp, but with the bug fixed. The code now uses post-increment (y--) to calculate 'res'. The code is as follows:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     // res=40
8     int y,u,o;
9     y=10;
10    u=5;
11    int res= y*--u;
12    cout<<"the res "<<res;
13
14    return 0;
15 }
16
```

The output of this code is now 40, as intended.



Operator Precedence		
<b>1</b>	<b>! Logical not</b>	<b>(Highest)</b>
<b>2</b>	<b>( ) Parenthesis</b>	
<b>3</b>	<b>* , / , %</b>	
<b>4</b>	<b>+ , -</b>	
<b>5</b>	<b>&gt; , &gt;= , &lt; , &lt;=</b>	
<b>6</b>	<b>== , !=</b>	
<b>7</b>	<b>&amp;&amp; (AND)</b>	
<b>8</b>	<b>   (OR)</b>	
<b>9</b>	<b>=</b>	<b>(Lowest)</b>

# Arithmetic Expressions

- ▶ Use spacing to make expressions readable
  - ▶ Which is easier to read?

$x+y*z$       or       $x + y * z$

- ▶ Precedence rules for operators are the same as used in your algebra classes
- ▶ Use parentheses to alter the order of operations

$x + y * z$  (y is multiplied by z first)

$(x + y) * z$  (x and y are added first)

Write c++ program that ask the user to enter 3 numbers and then program compute the average of these number and print the result on screen?

The screenshot shows the Code::Blocks IDE interface. On the left, the code editor window titled "main.cpp" displays the following C++ code:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int x,y,z;
8     cout<<"Enter three number to compute the average";
9     cin>>x>>y>>z;
10    int avg=(x+y+z)/3;
11    cout<<"the average is : "<<avg;
12
13    return 0;
14 }
```

On the right, the terminal window shows the output of the program execution:

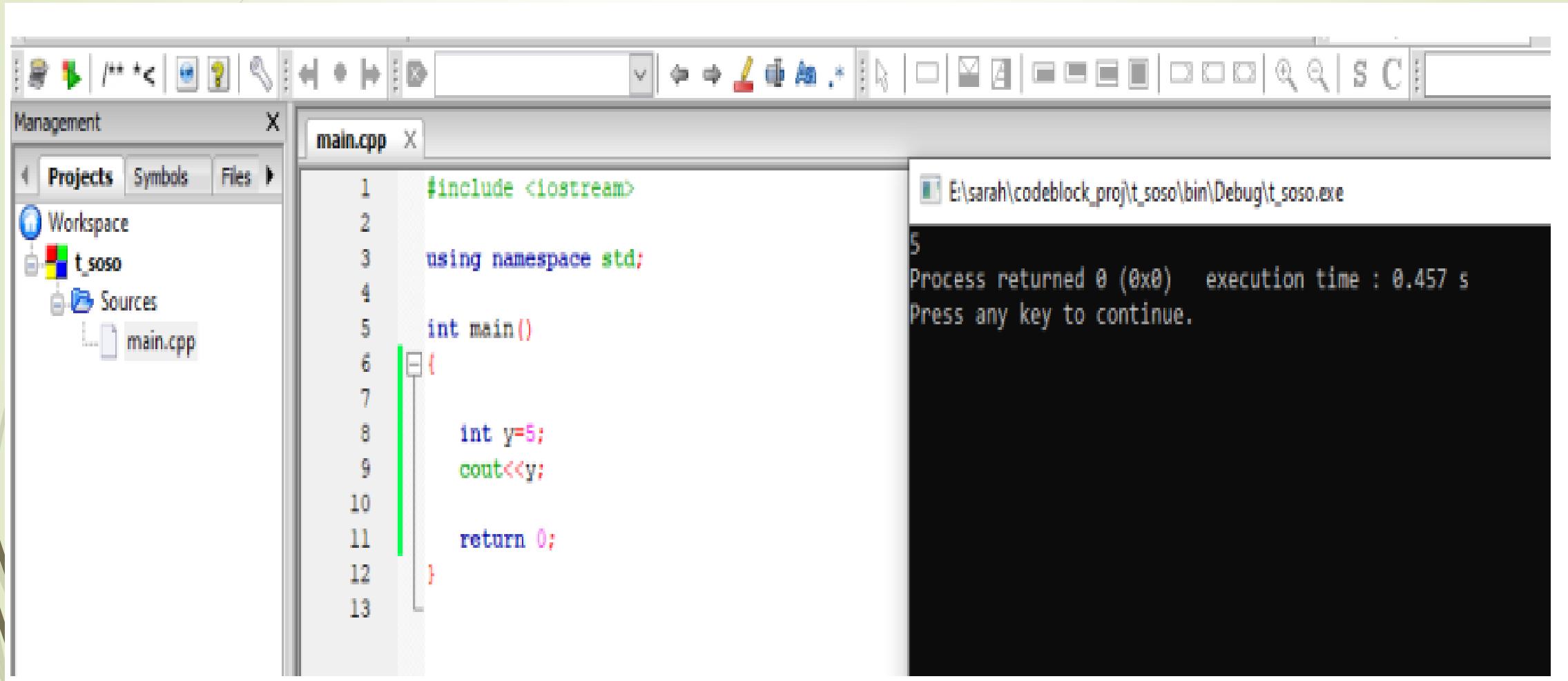
```
E:\sarah\codeblock_proj\prog_App\bin\Debug\prog_App.exe
Enter three number to compute the average 20 100 55
the average is : 58
Process returned 0 (0x0)  execution time : 9.378 s
Press any key to continue.
```

# Operator Shorthand(Assignment)

Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

In the example below, we use the assignment operator (=) to assign the value 5 to a variable called x:



The screenshot shows the Code::Blocks IDE interface. On the left, the Project Manager window displays a workspace named 't\_soso' containing a source file 'main.cpp'. The main editor window shows the following C++ code:

```
#include <iostream>
using namespace std;
int main()
{
    int y=5;
    cout<<y;
    return 0;
}
```

The output window on the right shows the execution results:

```
E:\sarah\codeblock_proj\t_soso\bin\Debug\t_soso.exe
5
Process returned 0 (0x0) execution time : 0.457 s
Press any key to continue.
```

# The addition assignment operator (+=) adds a value to a variable:

The screenshot shows the Code::Blocks IDE interface. On the left is the 'Management' window with tabs for 'Projects', 'Symbols', and 'Files'. A project named 't\_soso' is selected, with 'Sources' and 'main.cpp' listed. The main workspace contains a code editor window titled 'main.cpp' and a terminal window.

**Code Editor (main.cpp):**

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int x = 10;
8     x += 5;
9     cout << x;
10
11    return 0;
12 }
```

**Terminal Output:**

```
E:\sarah\codeblock_proj\t_soso\bin\Debug\t_soso.exe
15
Process returned 0 (0x0) execution time : 0.288 s
Press any key to continue.
```

A callout box points from the text 'X+=5 means X=X+5' in the terminal output to the line 'x += 5;' in the code editor.

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

int main() {
    int x = 5;
    x *= 3;
    cout << x;
    return 0;
}
```

15

### Example 3:

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

int main() {
    double x = 5;
    x /= 3;
    cout << x;
    return 0;
}
```

1.66667

# Comparison Operators

- In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.
- Comparison operators are used to compare two values. Note: The return value of a comparison is either true (1) or false (0).

A list of all comparison operators:

Operator	Name	Example
<code>==</code>	Equal to	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>&gt;</code>	Greater than	<code>x &gt; y</code>
<code>&lt;</code>	Less than	<code>x &lt; y</code>
<code>&gt;=</code>	Greater than or equal to	<code>x &gt;= y</code>
<code>&lt;=</code>	Less than or equal to	<code>x &lt;= y</code>

## Example1:

The screenshot shows the Code::Blocks IDE interface. On the left, the Projects panel displays a workspace with a project named "t\_soso" containing a source file "main.cpp". The main window shows the source code for "main.cpp". The code includes an `#include <iostream>`, a `using namespace std;` directive, and a `int main()` function. Inside the main function, there is a comparison `x == y` which is false (5 is not equal to 3). The output window on the right shows the program's output: "0" followed by a message about the execution time.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int x = 5;
8     int y = 3;
9     cout << (x == y); // returns 0 (false) because 5 is not equal to 3
10
11     return 0;
12 }
```

E:\sarah\codeblock\_proj\t\_soso\bin\Debug\t\_soso.exe  
0  
Process returned 0 (0x0) execution time : 0.292 s  
Press any key to continue.

## Example2:

The screenshot shows the Code::Blocks IDE interface. On the left, the project tree is visible with a workspace named 't\_soso' containing a source folder 'Sources' and a file 'main.cpp'. The central area displays the code for 'main.cpp':

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

int main()
{
    int x = 5;
    int y = 3;
    cout << (x != y); // returns 0 (false) because 5 is not equal to 3

    return 0;
}
```

The right side shows the terminal window with the output of the program:

```
E:\sarah\codeblock_proj\t_soso\bin\Debug\t_soso.exe
1
Process returned 0 (0x0) execution time : 1.004 s
Press any key to continue.
```

## Example 3

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

int main() {
    int x = 5;
    int y = 3;
    cout << (x > y); // returns 1 (true) because 5 is greater than 3
    return 0;
}
```

```
1
```

## main.cpp



Run

```
4 int main() {
5     int a, b;
6     a = 3;
7     b = 5;
8     bool result;
9
10    result = (a == b); // false
11    cout << "3 == 5 is " << result << endl;
12
13    result = (a != b); // true
14    cout << "3 != 5 is " << result << endl;
15
16    result = a > b; // false
17    cout << "3 > 5 is " << result << endl;
18
19    result = a < b; // true
20    cout << "3 < 5 is " << result << endl;
21
22    result = a >= b; // false
23    cout << "3 >= 5 is " << result << endl;
24
25    result = a <= b; // true
26    cout << "3 <= 5 is " << result << endl;
27
```

## Output

```
/tmp/TMLJL7aFRY.o
3 == 5 is 0
3 != 5 is 1
3 > 5 is 0
3 < 5 is 1
3 >= 5 is 0
3 <= 5 is 1
```

# Logical Operators

Operator	Name	Description	Example
<code>&amp;&amp;</code>	Logical and	Returns true if both statements are true	<code>x &lt; 5 &amp;&amp; x &lt; 10</code>
<code>  </code>	Logical or	Returns true if one of the statements is true	<code>x &lt; 5    x &lt; 4</code>
<code>!</code>	Logical not	Reverse the result, returns false if the result is true	<code>!(x &lt; 5 &amp;&amp; x &lt; 10)</code>

# Example for And &&

```
#include <iostream>

using namespace std;

int main()
{
    int y=10;
    int v=-5;
    if( y>0 &&v>0 ){

        cout<< y/v;
    }
    else{

        cout<<" there is a negative value";
    }

    return 0;
}
```

```
E:\sarah\codeblock_proj\eee\bin\Debug\eee.exe
there is a negative value
Process returned 0 (0x0) execution time : 0.174 s
Press any key to continue.
```

# Example :Or ||

main.cpp X

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     string grade;
8     int score;
9     cout<<"enter grade and score" << endl;
10    cin>>grade >>score;
11
12    if(grade=="good" || score>=65) {
13        cout<<"you passed and go to the next step";
14    }
15    else{
16
17        cout<<"please try again";
18    }
19
20    return 0;
21}
22
```

E:\sarah\codeblock\_pro\eee\bin\Debug\eee.exe

```
enter grade and score
good 70
you passed and go to the next step
Process returned 0 (0x0) execution time : 13.834 s
Press any key to continue.
```

# Example for Not

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7
8     int num1,num2;
9     float res;
10    cout<<"enter num1 and num2"<<endl;
11    cin>>num1>>num2 ;
12
13    if((num1>=10) && !(num2==0)) {
14        res=num1/num2;
15        cout<<res;
16    }
17    else{
18
19        cout<<"retype num2 not equal zero";
20    }
21
22    return 0;
23
24 }
```

```
E:\sarah\codeblock_proj\eee\bin\Debug\eee.exe
enter num1 and num2
10 5
2
Process returned 0 (0x0) execution time : 5.899 s
Press any key to continue.
```

# This also right

The screenshot shows a Code::Blocks IDE interface. On the left, the code editor window titled "main.cpp" displays the following C++ code:

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7
8     float num1, num2;
9     float res;
10    cout<<"enter num1 and num2"<<endl;
11    cin>>num1>>num2 ;
12
13    if(num1>=10 && !num2==0) {
14        res=num1/num2;
15        cout<<res;
16    }
17    else{
18
19        cout<<"retype num2 not equal zero";
20    }
21
22    return 0;
23 }
```

On the right, the terminal window titled "E:\sarah\codeblock\_proj\eee\bin\Debug\eee.exe" shows the program's output:

```
enter num1 and num2
10
6
1.66667
Process returned 0 (0x0) execution time : 2.859 s
Press any key to continue.
```

# AND

- ▶ Boolean expressions can be combined into more complex expressions with
  - ▶ `&&` -- The AND operator
    - ▶ True if both expressions are true
  - ❖ Syntax: `(Comparison_1) && (Comparison_2)`
- ▶ Example: `if ( (2 < x) && (x < 7) )`
  - ❑ True only if `x` is between 2 and 7
  - ❑ Inside parentheses are optional but enhance meaning

# OR

- ▶ || :The OR operator (no space!)
  - ▶ True if either or both expressions are true
- ▶ Syntax: (Comparison\_1) || (Comparison\_2)
- ▶ Example: if ( (x == 1) || (x == y) )
  - True if x contains 1
  - True if x contains the same value as y
  - True if both comparisons are true

# NOT

- $!$  -- negates any Boolean expression
- $!(x < y)$ 
  - True if x is NOT less than y
- $!(x == y)$ 
  - True if x is NOT equal to y
- ! Operator can make expressions difficult to understand...use only when appropriate

# Inequalities

- Be careful translating inequalities to C++
- if  $x < y < z$  translates as

`if ( ( x < y ) && ( y < z ) )`

NOT

`if ( x < y < z )`



```
int main()
{
    int y=5;
    int c=33, z=10;
    if(y<c && c<z ) {

        cout<<"true:";

    }
    else{

        cout<<"false:";

    }

    return 0;
}
```

# Example

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

int main() {
    int a, b;

    // 2 is assigned to a
    a = 2;

    // 7 is assigned to b
    b = 7;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "\nAfter a += b;" << endl;

    // assigning the sum of a and b to a
    a += b; // a = a +b
    cout << "a = " << a << endl;

    return 0;
}
```

## Output

a = 2

b = 7

After a += b;

a = 9

# Conditional operator ( ? )

- The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:

```
condition ? result1 : result2
```

If condition is true the expression will return result1, if it is not it will return result2.

```
7==5 ? 4 : 3      // returns 3, since 7 is not equal to 5.  
7==5+2 ? 4 : 3    // returns 4, since 7 is equal to 5+2.  
5>3 ? a : b       // returns the value of a, since 5 is greater than 3.  
a>b ? a : b       // returns whichever is greater, a or b.
```



```
// conditional operator

#include <iostream>
using namespace std;

int main ()
{
    int a,b,c;

    a=2;
    b=7;
    c = (a>b) ? a : b;

    cout << c;

    return 0;
}
```

# C++ Program to Find the Size of a Data Types

```
int main() {  
  
    int integerType;  
    char charType;  
    float floatType;  
    double doubleType;  
  
    // Calculate and Print  
    // the size of integer type  
    cout << "Size of int is: " << sizeof(integerType)  
        << "\n";  
  
    // Calculate and Print  
    // the size of doubleType  
    cout << "Size of char is: " << sizeof(charType) << "\n";  
  
    // Calculate and Print  
    // the size of charType  
    cout << "Size of float is: " << sizeof(floatType)  
        << "\n";  
  
    // Calculate and Print  
    // the size of floatType  
    cout << "Size of double is: " << sizeof(doubleType)  
        << "\n";  
  
    return 0;  
}
```

```
D:\app\dody\bin\Debug\dody.exe  
Size of int is: 4  
Size of char is: 1  
Size of float is: 4  
Size of double is: 8
```

```
Process returned 0 (0x0)  executi  
Press any key to continue.
```

# References

- ▶ Sharanya Jayaraman, introduction to C++ ,  
Department of Computer Science Florida State  
University 2024.
- ▶ Juan Soulié,C++ Language Tutorial 2008, Available  
online at : <http://www.cplusplus.com/doc/tutorial/>



**Thank you**