University of Mosul

College of Computer Sciences and Mathematics

Department of Artificial Intelligence

Algorithms and Structured Programming (I)

الخوارزميات والبرمجة المهيكلة (1)

First stage

Lecture 3

أ.م. بيداء سليمان بهنام

Assist.Prof. Baydaa Sulaiman Bahnam

Lecture 3 Outline

1. Operators in C++
2. Types of Operators in C++
3. Arithmetic operators
4. Relational operators
5. Logical operators
6. Assignment Operators
7. Unary Operators
8. Bitwise Operator
9. Ternary Operator
10. Other Special Operators
11. Examples

Operators in C++

C++ provides a wide range of operators, which are the symbols that operate on values to perform specific mathematical or logical computations on given values. They are the foundation of any programming language.

Types of Operators in C++

C++ operators are classified into 8 types as follows::

1.Arithmetic Operators
2.Relational Operators
3.Logical Operators
4.Assignment Operators
5.Unary Operators
6.Bitwise Operators
7.Ternary Operator
8.Special Operators

1. Arithmetic operators

Arithmetic operators perform mathematical calculations. x=10, y=5

Operator	Description	Example	Result	Explanation
`+`	Addition	`x + y`	15	Adds `x` and `y`.
`-`	Subtraction	`x - y`	5	Subtracts `y` from `x`.
`*`	Multiplication	`x * y`	50	Multiplies `x` by `y`.
`/`	Division	`x / y`	2	Divides `x` by `y`.
`%`	Modulus (Remainder)	`x % y`	1	Finds the remainder of `x/y`.

Write a C++ program that demonstrates the use of **arithmetic operators** of two numbers (num1 and num2) for performing sum, subtraction, multiplication, division, and modulus (remainder) operations.

```
Enter the first number: 10
Enter the second number: 3

Results of arithmetic operations:
Sum: 13
Subtraction: 7
Multiplication: 30
Division: 3
Modulus(Remainder): 1
```

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;
    // Perform arithmetic operations
    int sum = num1 + num2;
    int subtraction = num1 - num2;
    int multiplication = num1 * num2;
    int division = num1 / num2;
    int remainder = num1 % num2;

    // Display the results
    cout << "\nResults of arithmetic operations:\n";
    cout << "Sum: " << sum << endl;
    cout << "Subtraction: " << subtraction << endl;
    cout << "Multiplication: " << multiplication << endl;
    cout << "Division: " << division << endl;
    cout << "Modulus(Remainder): " << remainder << endl;
    return 0;
}
```

2. Relational (Comparison) operators

These operators compare two values and return a Boolean result (true or false).

Operator	Description	Example	Returns
`==`	Equal to	`a == b`	`true` if `a` is equal to `b`, otherwise `false`
`!=`	Not equal to	`a != b`	`true` if `a` is not equal to `b`, otherwise `false`
`<`	Less than	`a < b`	`true` if `a` is less than `b`, otherwise `false`
`<=`	Less than or equal to	`a <= b`	`true` if `a` is less than or equal to `b`, otherwise `false`
`>`	Greater than	`a > b`	`true` if `a` is greater than `b`, otherwise `false`
`>=`	Greater than or equal to	`a >= b`	`true` if `a` is greater than or equal to `b`, otherwise `false`

Ex: Assume a=5, b= 10

Expression	Description	Result	Boolean Value	Explanation
`a == b`	Is `a` equal to `b`?	`0`	`false`	`5` is not equal to `10`
`a != b`	Is `a` not equal to `b`?	`1`	`true`	`5` is not equal to `10`
`a > b`	Is `a` greater than `b`?	`0`	`false`	`5` is not greater than `10`
`a < b`	Is `a` less than `b`?	`1`	`true`	`5` is less than `10`
`a >= b`	Is `a` greater than or equal to `b`?	`0`	`false`	`5` is not greater than or equal to `10`
`a <= b`	Is `a` less than or equal to `b`?	`1`	`true`	`5` is less than or equal to `10`

Relational (Comparison) operators

Example of Ascii values comparison

Operator	Description	Example	ASCII Value Comparison	Returns	Explanation
`==`	Equal to	`'A' == 'A'`	`65 == 65`	`true`	Both characters have the same ASCII value.
`!=`	Not equal to	`'A' != 'B'`	`65 != 66`	`true`	Characters have different ASCII values.
`<`	Less than	`'A' < 'B'`	`65 < 66`	`true`	ASCII value of 'A' is less than 'B'.
`<=`	Less than or equal to	`'A' <= 'A'`	`65 <= 65`	`true`	ASCII value of 'A' is equal to 'A'.
`>`	Greater than	`'B' > 'A'`	`66 > 65`	`true`	ASCII value of 'B' is greater than 'A'.
`>=`	Greater than or equal to	`'B' >= 'A'`	`66 >= 65`	`true`	ASCII value of 'B' is greater than or equal to 'A'.

ASCII Table

Decimal	Character	Decimal	Character	Decimal	Character	Decimal	Character
48	'0'	65	'A'	97	'a'	38	'&'
49	'1'	66	'B'	98	'b'	42	'*'
50	'2'	67	'C'	99	'c'	43	'+'
51	'3'	68	'D'	100	'd'	47	'/'
52	'4'	69	'E'	101	'e'	58	':'
53	'5'	70	'F'	102	'f'	59	';'
54	'6'	71	'G'	103	'g'	60	'<'
55	'7'	72	'H'	104	'h'	61	'='
56	'8'	73	'I'	105	'i'	62	'>'
57	'9'	74	'J'	106	'j'	63	'?'
...	...	...	...	...	...	...	...
88	'X'	120	'x'				
89	'Y'	121	'y'				
90	'Z'	122	'z'				

This table includes the following:

- Numbers (48-57 for '0' to '9')
- Uppercase letters (65-90 for 'A' to 'Z')
- Lowercase letters (97-122 for 'a' to 'z')
- Common symbols

3. Logical operators

Logical operators are used to perform logical operations and return Boolean results.

AND (&&) Table		
A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

AND (&&) Table		
A	B	A&&B
false	false	false
false	true	false
true	false	false
true	true	true

OR (||) Table		
A	B	A||B
0	0	0
0	1	1
1	0	1
1	1	1

OR (||) Table		
A	B	A||B
false	false	false
false	true	true
true	false	true
true	true	true

NOT (!) Table	
A	!A
0	1
1	0

NOT (!) Table	
A	!A
false	true
true	false

Logical Operators		
For all examples below consider a = 10 and b = 5		
Operator	**Description**	**Example**
&&	Logical AND	(a>b) && (b==5) gives true
||	Logical OR	(a>b) || (b==2) gives true
!	Logical NOT	!(b==5) gives false

Find the result

Ex. What is the final result of variable *z* after executing all of the following statements?

```
int  x = 9, y = 9;
int d = 4, f = 4;

int z = (x == y) && (d == f);
z = 1 && 1;    //   z = 1
```

What is the final result of variable *z* if we change the value of f =4 and d=3?

```
z = (x == y) && (d == f);
z=  1&&0  ;       //   z=0
```

What is the final result of variable *z* if we change the && operator to ||?

```
z = (x == y) || (d == f);
z= 1  || 0          //    z=1
```

Ex. What is the result of variable z after executing each of the following statements step by step?

```
int x=15, y=6, z;
z= x==y;         //  z=0
z= !(x==y);      //  z=1
z= x>y;          //   z=1
z= !(x>y);       //  z=0
```

Ex. What is the result of variable z after executing each of the following statements step by step? If you know x=15 , y=6

1.

```
z= (x==y)&&(x>y);    z= 0&&1    //  z=0
z= (x==y)&&(x<y);    z= 0&&0    //  z=0
z= (x!=y)&&(x<y);    z= 1&&0    //  z=0
z= (x!=y)&&(x>y);    z= 1&&1    //  z=1
```

2.

```
z= (x==y)||(x>y);    z= 0||1    //  z=1
z= (x==y)||(x<y);    z= 0||0    //  z=0
z= (x!=y)||(x<y);    z= 1||0    //  z=1
z= (x!=y)||(x>y);    z= 1||1    //  z=1
```

3.

```
bool a = true, b = false;
cout << (a && b) << endl; // Output: 0 (false)
cout << (a || b) << endl; // Output: 1 (true)
cout << (!a) << endl;     // Output: 0 (false)
```

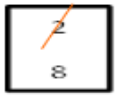
4. Assignment Operators

These operators assign values to variables.

Operator	Description	Example	Example
=	Assignment	a = 10	a = 10
+=	Add and assign	a += 5	a = a + 5
-=	Subtract and assign	a -= 3	a = a - 3
*=	Multiply and assign	a *= 2	a = a * 2
/=	Divide and assign	a /= 2	a = a / 2
%=	Modulus and assign	a %= 2	a = a % 2

وظيفة هذه المعاملات أنها تقوم بعملية حسابية رياضية مع اسناد في وقت واحد.

Ex: int a=5, b=2;
a=b ; b=8;




```
int x = 10;
x = x + 6;    /* x = 10 + 6 */
cout << x;  /* 16 */
```

نفس الكود السابقة لكن باستخدام المعامل (=+) , وسيكون كالتالي:

```
int x = 10;
x += 6;    /* x = x+6 = 10+6 */
cout << x;  /* 16 */
```

Ex:

```
int i = 9;
i /= 3;  /* i = i/3 = 9/3 = 3 */
```

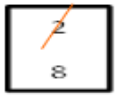
4. Assignment Operators

These operators assign values to variables.

Operator	Description	Example	Example
=	Assignment	a = 10	a = 10
+=	Add and assign	a += 5	a = a + 5
-=	Subtract and assign	a -= 3	a = a - 3
*=	Multiply and assign	a *= 2	a = a * 2
/=	Divide and assign	a /= 2	a = a / 2
%=	Modulus and assign	a %= 2	a = a % 2

وظيفة هذه المعاملات أنها تقوم بعملية حسابية رياضية مع اسناد في وقت واحد.

Ex: int a=5, b=2;
a=b ; b=8;




```
int x = 10;
x = x + 6;    /* x = 10 + 6 */
cout << x;  /* 16 */
```

نفس الكود السابقة لكن باستخدام المعامل (=+) , وسيكون كالتالي:

```
int x = 10;
x += 6;    /* x = x+6 = 10+6 */
cout << x;  /* 16 */
```

Ex:

```
int i = 9;
i /= 3;  /* i = i/3 = 9/3 = 3 */
```

5. Unary Operators

Unary operators operate on a single operand. أن هذه المعاملات تُستخدم مع قيمة واحدة فقط (أو متغير واحد) لإجراء عملية معينة عليها

Operator	Description	Example
`+`	Unary plus (positive sign)	`+a`
`-`	Unary minus (negate value)	`-a`
`!`	Logical NOT (negate boolean)	`!a`
`++`	Increment	`++a` or `a++`
`--`	Decrement	`--a` or `a--`
`&`	Address-of	`&a`
`*`	Dereference	`*ptr`

I. Unary plus (positive sign) Operators معامل الجمع الاحادي

The symbol (+) represents the unary plus operator in C++, which explicitly specifies the positive value of its operand. It doesn't change the sign of x if it's already positive, but it can be used for clarity in expressions.

يمثل الرمز (+) معامل الجمع الأحادي والذي يحدد بوضوح القيمة الموجبة لمعامله. ولا يغير هذا الرمز علامة المتغير (x مثلا) إذا كانت سالبة او موجبة بالفعل، ولكن يمكن استخدامه للتوضيح في التعبيرات.

```
#include <iostream>
using namespace std;  // Use the standard namespace
void main()
{
    // Unary plus (positive sign)
    int x = -5;
    int y = +x;  // Unary plus, y stays as -5, because + doesn't change the sign
of a negative number
    cout << "Unary plus (negative): " << y << endl;  // Output: -5

    x = 5;
    y = +x;  // Unary plus, y stays as 5, because it's already positive
    cout << "Unary plus (positive): " << y << endl;  // Output: 5
}
```

II. Unary Minus (-) Operator معامل الطرح الأحادي (−)

The unary minus operator is used to change the sign of an operand. It changes a positive operand to negative and a negative operand to positive. For example:

Find the inverse (or negate) value of x.

```
Ex1.// Unary minus (negate value)
    x = 5;
    y = -x;  // Unary minus, y becomes -5
    cout << "Unary minus: " << y << endl;  // Output: -5

Ex2.// Unary minus (negate value)
    x = -10;
    y = -x;  // Unary minus, y becomes 10
    cout << "Unary minus: " << y << endl;  // Output: 10
```

Difference between minus(-) unary operator and minus(-) arithmetic operator:

The minus (-) unary operator and minus(-) arithmetic operator look same, however they are completely different. The unary – operator works on a single operand while the arithmetic – operator works on two operands. For example:

int num1 = 10, num2 = 5, num3 = 100, sub;

//this is minus(-) arithmetic operator

//working on two operands num1 and num2

sub = num1 - num2;

//this is minus (-) unary operator

//working on a single operand

int inverseNum3 = -num3; **//value of inverseNum3 is -100**

III. Logical NOT (negate Boolean) Operator

The logical NOT operator (!) negates a Boolean expression. If the value is true, it becomes false, and vice versa.

```
#include <iostream>
using namespace std;
int main()
{
    bool z = false; // Variable z is initialized to false
    cout << "Original value of z: " << z << endl;
    cout << "Negated value of z: " << !z  << endl;
    return 0;
}
```

```
Original value of z: 0
Negated value of z: 1
```

IV. Increment Operators

V. Decrement Operators

- **Postfix Increment** (`a++`): This is when the increment operator is placed **after** the variable (e.g., `a++`). It uses the current value of the variable in an expression **before** incrementing it by 1. This is why it's called **postfix** increment.

a ++ تعني نستخدم a في التعبير الحسابي ثم يتم زيادها بواحد. اي الزياد بواحد بعد الاستخدام في التعبير الحسابي.

- **Postfix Decrement** (`b--`): This is when the decrement operator is placed **after** the variable (e.g., `b--`). It uses the current value of the variable in an expression **before** decrementing it by 1. This is why it's called **postfix** decrement.

b-- تعني نستخدم b في التعبير الحسابي ثم يتم نقصانها بواحد. اي النقصان بواحد بعد الاستخدام في التعبير الحسابي.

- **Prefix Increment** (`++a`): This is when the increment operator is placed **before** the variable (e.g., `++a`). It increases the variable's value by 1 **before** using it in an expression. This is why it's called **prefix** increment.

++a تعني زيادة a بواحد ثم استخدامها في التعبير الحسابي. اي الزيادة بواحد قبل الاستخدام في التعبير الحسابي.

- **Prefix Decrement** (`--b`): This is when the decrement operator is placed **before** the variable (e.g., `--b`). It decreases the current value of the variable by 1 **before** using it in an expression. This is why it's called **prefix** decrement.

--b تعني نقصان b بواحد ثم استخدامها في التعبير الحسابي. اي النقصان بواحد قبل الاستخدام في التعبير الحسابي.

```
a++;
++a;
```

a=a+1;

تعني الزيادة بمقدار واحد ويمكن كتابتها بصورة مكافئة على النحو التالي :

```
b--;
--b;
```

b=b-1;

تعني النقصان بمقدار واحد ويمكن كتابتها بصورة مكافئة على النحو التالي :

Increment and decrement operators

Assume a=10 , find the values of *a* and *result* in the following:

Operation	Behavior	Example (`a = 10`)	Final `a`	Result Value
`a++`	Use `a`, then increment it.	`result = a++;`	11	10
`++a`	Increment `a`, then use the incremented value.	`result = ++a;`	11	11
`a--`	Use `a`, then decrement it.	`result = a--;`	9	10
`--a`	Decrement `a`, then use the decremented value.	`result = --a;`	9	9

Explanation:

- `a++` **(Post-increment):**
  - The current value of `a` (which is 10) is used in `result`, and then `a` is incremented to 11.
- `++a` **(Pre-increment):**
  - `a` is first incremented to 11, and then the new value of `a` is assigned to `result`.
- `a--` **(Post-decrement):**
  - The current value of `a` (which is 10) is used in `result`, and then `a` is decremented to 9.
- `--a` **(Pre-decrement):**
  - `a` is first decremented to 9, and then the new value of `a` is assigned to `result`.

Ex

```
#include <iostream>
using namespace std;
int main()
{
    int a=5,b=10;
    a = a++;
    b = b--;
    cout << "  The value of a  after a++  = " << a <<"\n";
    cout << "  The value of b after b--  = " << b;
    return 0;
}
```

```
The value of a  after a++  = 6
The value of b after b--  = 9
```

ملاحظة :هنا لا نلاحظ الفرق في الناتج في حالة استخدام الزيادة او النقصان بمقدار واحد لكل واحدة من (a++,b--,++a,--b) عند وضع الناتج بنفس اسم المتغير وعلى سطر واحد (وبدون اي تعبير حسابي لمتغيرات اخرى)

Ex

```
#include <iostream>
using namespace std;
int main()
{
    int a=5,b=10;
    a = ++a;
    b = --b;
    cout << "  The value of a  after ++a  = " << a <<"\n";
    cout << "  The value of b after --b  = " << b;
    return 0;
}
```

```
The value of a  after a++  = 6
The value of b after b--  = 9
```

Ex **1. Postfix Increment (a++)**

ملاحظة: هنا نلاحظ الفرق في الناتج في حالة استخدام الزيادة او النقصان بمقدار واحد لكل واحدة من (a++,b--,++a,--b) عندما وضع الناتج باسم متغير اخر (اي يوجد تعبير حسابي لمتغيرات اخرى).

```
int a = 5;

int result = a++; // result is assigned 5, then a is incremented to 6
                  /* يجعل result يساوي a ثم يزيد a بواحد */
```

2. Postfix Decrement (b--)

```
int b = 5;

int result = b--; // result is assigned 5, then b is decremented to 4
                  /* يجعل result يساوي b ثم ينقص b بواحد */
```

3. Prefix Increment (++a)

```
int a = 5;

int result = ++a; // a is incremented to 6, then result is assigned 6
                  /* يزيد a بواحد أولاً ثم يضع قيمته في result */
```

4. Prefix Decrement (--b)

```
int b = 5;

int result = --b; // b is decremented to 4, then result is assigned 4
                  /* ينقص b بواحد أولاً ثم يضع قيمته في result */
```

Ex. Find the value of a and b :

1). If a=5 , b=a++

Solution: b=5 a=6

2). If a=5 and b=++a

Solution : b=6 a=6

Ex. Find value of i and j :

1). If i=7 , j=4+ --i; **Solution :i=6 j=4+6=10**

2). If i=7 , j=4+ i--; **Solution: j=11 i=6**

Ex: What is the result of variables x,y,z after executing each of the following statements step by step?

```
main()
 {
       int x ,y ,z;
      x=y=z=0;
      x= ++y  +  ++z;          //   x=2 , y=1 , z=1
      x=y++  +  z++;           //   x=2  , y=2  , z=2
      x=++y  +  z++;           //   x=5  ,  y=3 , z=3

      x= y--  +  --z;          //   x=5  , y=2  , z=2
    cout << "x= " << x << "; y= " << y << "; z= " << z;
  }
```

Solution : x=5; y=2; z=2;

VI. Address-of **VII. Dereference**

6. Bitwise Operator

Bitwise operators perform operations at the bit level.

Operator	Description	Example
&	Bitwise AND	a & b
|	Bitwise OR	a | b
^	Bitwise XOR	a ^ b
~	Bitwise NOT	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

الفرق بين AND و OR و XOR و NOT في العمليات المنطقية Logical Operators و العمليات باستخدام Bitwise Operators :

1. **العمليات المنطقية (Logical Operators)**
   - تُستخدم مع القيم المنطقية (Boolean values)
   - تُطبق على مستوى القيمة الكاملة، وليس على مستوى البتات(Bits)
   - الناتج دائمًا true أوfalse بناءً على العلاقة بين المدخلات

2. **العمليات باستخدام Bitwise Operators**
   - تعمل على مستوى البتات (Bits) داخل الأعداد الصحيحة (integer types) وتتعامل مع التمثيل الثنائي (Binary representation)
   - تُطبق العمليات على كل بت (bit) في الرقمين بشكل مستقل وتنتج رقما جديدا يمثل النتيجة على مستوى البت.

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a = 5, b = 3;
    cout << "Values: a = " << a << ", b = " << b << endl;

    // Bitwise AND
    cout << "Bitwise AND (a & b): " << (a & b) << endl;

    // Bitwise OR
    cout << "Bitwise OR (a | b): " << (a | b) << endl;

    // Bitwise XOR
    cout << "Bitwise XOR (a ^ b): " << (a ^ b) << endl;

    // Bitwise NOT
    cout << "Bitwise NOT (~a): " << (~a) << endl;

    // Left Shift
    cout << "Left Shift (a << 1): " << (a << 1) << endl;

    // Right Shift
    cout << "Right Shift (a >> 1): " << (a >> 1) << endl;

    return 0;
}
```

```
Values: a = 5, b = 3
Bitwise AND (a & b): 1
Bitwise OR (a | b): 7
Bitwise XOR (a ^ b): 6
Bitwise NOT (~a): -6
Left Shift (a << 1): 10
Right Shift (a >> 1): 2
```

توضيح نتائج البرنامج عند تنفيذ عمليات bitwise على الـ bits

القيمة الثنائية (Binary)

Bitwise AND (a & b):

b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	1	0	1	5	a
0	0	0	0	0	0	1	1	3	b
0	0	0	0	0	0	0	1	1	a & b

Bitwise OR ($a|b$):

b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	1	0	1	5	a
0	0	0	0	0	0	1	1	3	b
0	0	0	0	1	1	1	7		a | b

Bitwise XOR ($a \hat{} b$):

b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	1	0	1	5	a
0	0	0	0	0	0	1	1	3	b
0	0	0	0	0	1	1	0	6	a^b

القيمة الثنائية (Binary)

Bitwise NOT :(~a)

b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	1	0	1	5	a
1	1	1	1	1	0	1	0	-6	~a

Left Shift :(a<<1)

b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	1	0	1	5	a
0	0	0	0	1	0	1	0	10	a<<1

Right Shift :(a>>1)

b7	b6	b5	b4	b3	b2	b1	b0		
0	0	0	0	0	1	0	1	5	a
0	0	0	0	0	0	1	0	2	(a>>1)

7. Ternary Operator (Conditional Operator) اداة الشرط ?:

The ternary operator is a shorthand for an if-else statement.

Operator	Description	Example
`? :`	Conditional (ternary) operator	`a > b ? x : y`

اداة الشرط `?:` يقال له Conditional أو Ternary Operator لأنه يأخذ ثلاث عناصر ليعمل.

يمكن إستعماله بدل جمل الشرط `if` و `else` في حال كنت تريد إعطاء قيمة للمتغير.

The syntax is:

variable = (condition) ? result1 : result2 ;

معناها انه يتم تنفيذ النتيجة الاولى result1 عندما يكون جواب الشرط true والا فيتم تنفيذ النتيجة الثانية result2 اي يكون جواب الشرط false

```
if(condition)
  result1;
else
  result2;
```

Ex 1:

```
int main()
{
    int a = 5, b;
    b = a > 1 ? 10 : 20; // Ternary operator
    cout << b << endl;  // The output b = 10
    return 0;
}
```

Ex2:

```
int main()
 {
    int x, y;
    cin >> x;
    y = x >= 0 ? x : -x;
    cout << y << endl;
    return 0;
}
```

If you enter x=4 then y=4, but if you enter x= -7 then y=7

8. Other Special Operators

Operator	Description	Example
`sizeof`	Returns the size of a variable	`sizeof(int)`
`typeid`	Returns type information	`typeid(a).name()`
`new`	Allocates memory dynamically	`int* ptr = new int;`
`delete`	Deallocates memory	`delete ptr;`

Ex.

```
int a = 5;
cout << "Size of int: " << sizeof(a) << " bytes" << endl;
```

```
Size of int: 4 bytes
```

sizeof

The **sizeof** operator in C++ is used to determine the size (in bytes) of a data type or object in memory. It is a compile-time operator, meaning it evaluates the size of the data type or variable during the compilation phase.

Syntax:

```
sizeof(expression)
sizeof(data_type)
```

- `expression`: Can be a variable, object, or data type.
- `data_type`: Specifies the type for which the size is being calculated.

```
Size of int: 4 bytes
Size of x: 4 bytes
Size of double: 8 bytes
Size of y: 8 bytes
Size of char: 1 bytes
Size of z: 1 bytes
```

```cpp
#include <iostream>
using namespace std;
void main()
{
    int x = 42;
    double y = 3.14;
    char z = 'A';
    cout << "Size of int: " << sizeof(int) << " bytes" << endl;  // Size of int data type
    cout << "Size of x: " << sizeof(x) << " bytes" << endl;      // Size of variable x
    cout << "Size of double: " << sizeof(double) << " bytes" << endl;
    cout << "Size of y: " << sizeof(y) << " bytes" << endl;
    cout << "Size of char: " << sizeof(char) << " bytes" << endl;
    cout << "Size of z: " << sizeof(z) << " bytes" << endl;
}
```