**QUE 1 : What are the different data types available in C++? Explain with examples.**

**1. Basic Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Meaning** | **Example** |
| int | Integer numbers (no decimals) | int age = 20; |
| float | Decimal numbers (single precision) | float price = 12.5; |
| double | Decimal numbers (double precision, more accurate) | double distance = 12345.6789; |
| char | Single character | char grade = 'A'; |
| bool | Boolean (true/false) | bool isPassed = true; |

**2. Derived Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Meaning** | **Example** |
| Array | Collection of elements of same type | int marks[5] = {90, 80, 70, 85, 75}; |
| Pointer | Variable that stores memory address | int\* p; |
| Reference | Another name for a variable | int a = 10; int &ref = a; |

**3. User-Defined Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Meaning** | **Example** |
| struct | Group of variables under one name | struct Student {  int id;  char name[50];  float marks;  }; |

**QUE 2 : Explain the difference between implicit and explicit type conversion in C++.**

**1. Implicit Type Conversion (Type Promotion)**

* **Also called:** *Automatic Type Conversion*.
* Happens **automatically** by the compiler.
* Converts a smaller type to a bigger type **without your help**.
* It is **safe** because there’s no data loss (most of the time).

**Example:**

#include <iostream>

using namespace std;

int main() {

int num = 10;

float f = num; // int automatically converted to float

cout << f; // Output: 10.0

return 0;

}

🔹 Here, int is **automatically converted** into float.

**2. Explicit Type Conversion (Type Casting)**

* **Also called:** *Manual Type Conversion*.
* You **forcefully** tell the compiler to change the data type.
* Used when you **want control** over how the conversion happens.

**Example:**

#include <iostream>

using namespace std;

int main() {

float pi = 3.14;

int intPi = (int)pi; // explicitly cast float to int

cout << intPi; // Output: 3

return 0;

}

🔹 Here, we **manually cast** pi to int using (int).

**Quick Comparison:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Implicit Conversion** | **Explicit Conversion** |
| Who does it? | Compiler | Programmer |
| How? | Automatically | Manually (using cast) |
| Example | int → float automatically | (int)floatVar manually |
| Risk | Low | Medium (possible data loss) |

**QUE 3 : What are the different types of operators in**

**C++? Provide examples of each.**

**1. Arithmetic Operators**

* Perform **mathematical operations**.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| + | Addition | 5 + 3 = 8 |
| - | Subtraction | 5 - 3 = 2 |
| \* | Multiplication | 5 \* 3 = 15 |
| / | Division | 6 / 3 = 2 |
| % | Modulus (remainder) | 5 % 3 = 2 |

Example:

int a = 10, b = 3;

cout << (a + b); // 13

**2. Relational (Comparison) Operators**

* Compare two values and return **true or false**.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| Equal | Equal to | 5 == 5 (true) |
| >= | Not equal to | 5 != 3 (true) |
| > | Greater than | 5 > 3 (true) |
| < | Less than | 5 < 3 (false) |
| >= | Greater than or equal to | 5 >= 5 (true) |
| <= | Less than or equal to | 3 <= 5 (true) |

Example:

int x = 10, y = 20;

cout << (x < y); // 1 (true)

**3. Logical Operators**

* Connect **multiple conditions**.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| && | Logical AND | (5 > 2) && (3 < 5) (true) |
| ` |  | ` |
| ! | Logical NOT | !(5 == 5) (false) |

Example:

bool a = true, b = false;

cout << (a && b); // 0 (false)

**4. Assignment Operators**

* Assign values to variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| = | Assign | x = 5; |
| += | Add and assign | x += 3; (same as x = x + 3;) |
| -= | Subtract and assign | x -= 2; |
| \*= | Multiply and assign | x \*= 2; |
| /= | Divide and assign | x /= 2; |
| %= | Modulus and assign | x %= 2; |

Example:

int x = 5;

x += 3; // x = 8

cout << x;

**5. Bitwise Operators**

* Perform operations on bits (0s and 1s).

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| & | Bitwise AND | 5 & 3 = 1 |
| ` | ` | Bitwise OR |
| ^ | Bitwise XOR | 5 ^ 3 = 6 |
| ~ | Bitwise NOT | ~5 = -6 |
| << | Left shift | 5 << 1 = 10 |
| >> | Right shift | 5 >> 1 = 2 |

Example:

int x = 5, y = 3;

cout << (x & y); // 1

**6. Unary Operators**

* Act on a **single operand**.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| ++ | Increment | ++x (adds 1) |
| -- | Decrement | --x (subtracts 1) |
| + | Positive sign | +x |
| - | Negative sign | -x |

Example:

int x = 5;

cout << ++x; // 6

**7. Ternary Operator**

* Short form of an **if-else**.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Meaning** | **Example** |
| ? : | Ternary | condition ? value\_if\_true : value\_if\_false |

Example:

int a = 5, b = 10;

int max = (a > b) ? a : b;

cout << max; // 10

**8. Other Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| sizeof | Size of a data type or variable | sizeof(int) |
| typeid | Gives type information | typeid(x).name() |
| & | Address of variable | &x |
| \* | Pointer to variable | \*ptr |

Example:

int x = 10;

cout << sizeof(x); // Typically 4

**Summary:**

|  |  |
| --- | --- |
| **Type** | **Examples** |
| Arithmetic | +, -, \*, /, % |
| Relational | ==, !=, <, > |
| Logical | &&, ` |
| Assignment | =, +=, -= |
| Bitwise | &, ` |
| Unary | ++, --, -, + |
| Ternary | ? : |
| Miscellaneous | sizeof, typeid, &, \* |

QUE 4 : Explain the purpose and use of constants and literals in C++.

**Constants in C++**

* A **constant** is a **fixed value** that **cannot be changed** once assigned.
* Used when you want a **value to stay the same** throughout the program.

**How to create a constant:**

* Use the const keyword.

**Example:**

const float PI = 3.14159;

Here, PI will always be 3.14159. You **cannot** change its value later.

**Literals in C++**

* **Literals** are **actual fixed values** written directly into the code.
* They represent **numbers, characters, strings, or boolean values**.

**Examples of literals:**

* 10 → Integer literal
* 3.14 → Floating-point literal
* 'A' → Character literal
* "Hello" → String literal
* true, false → Boolean literals

**Example in code:**

int age = 20; // 20 is an integer literal

char grade = 'A'; // 'A' is a character literal

float pi = 3.14; // 3.14 is a float literal

bool isPassed = true; // true is a boolean literal

**Difference:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Constants** | **Literals** |
| What? | Named fixed values | Actual fixed values directly written |
| Example | const int maxMarks = 100; | 100, 'A', "Hello" |
| Purpose | To use a **meaningful name** for a fixed value | To **write values directly** |

**Why use Constants and Literals?**

* **Constants** make programs **easier to read** and **maintain**.  
  (Instead of magic numbers like 3.14, you use PI.)
* **Literals** are needed whenever you **assign** or **compare** values directly.