Chapter 2

Introduction to C#

C# is a modern, object-oriented programming language developed by Microsoft as part of its .NET initiative. It is designed to be simple, powerful, and versatile, making it suitable for a wide range of applications.

Basic Features :

Object-Oriented: Supports abstraction, encapsulation, inheritance, and polymorphism.

Type-Safe: Prevents type errors by enforcing strict type rules.

Rich Standard Library: Provides a robust library for various functionalities.

Interoperable: Works seamlessly with other .NET languages.

Platform-Independent: Enables cross-platform development through .NET Core.

Memory Management: Automatic garbage collection and efficient memory handling.

Comments in c#

Comments are non-executable lines in code used to describe or explain the code's logic.

Purpose:

- Improve code readability.
- Help developers understand the code.
- Serve as documentation for future reference.

Types of Comments

Single-line Comments: Use // to comment a single line.

```
 // This is a single-line comment
```

Multi-line Comments: Use /* to start and */ to end a comment block.

```
/* This is a
   multi-line comment */
```

XML Documentation Comments: Use /// to generate documentation.

```
/// <summary>
/// This method adds two numbers.
/// </summary>
```

Variables

What are Variables?

- Containers for storing data.
- Each variable has a type that determines what kind of data it can hold.

Syntax

```
dataType variableName  = value ;
```

Example :

```
int age = 25;
string name = "John" ;
```

Data Types

What are Data Types?

- Define the type of data a variable can store.

Value Types: Store data directly (e.g., int, double, char, bool).

Reference Types: Store references to data (e.g., string, arrays, objects).

Common Value Types:

int: Whole numbers.

double: Decimal numbers.

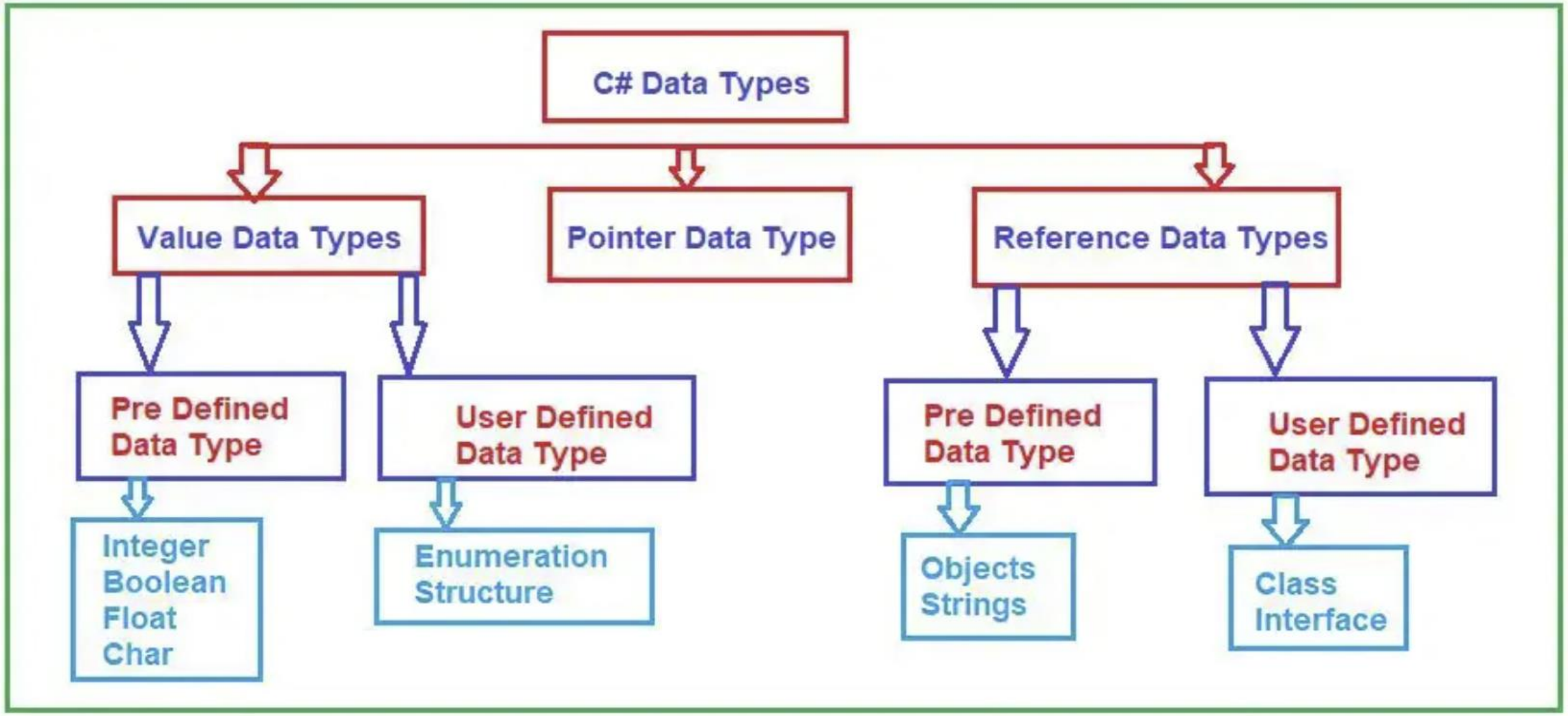
char: Single characters.

bool: True/false values.

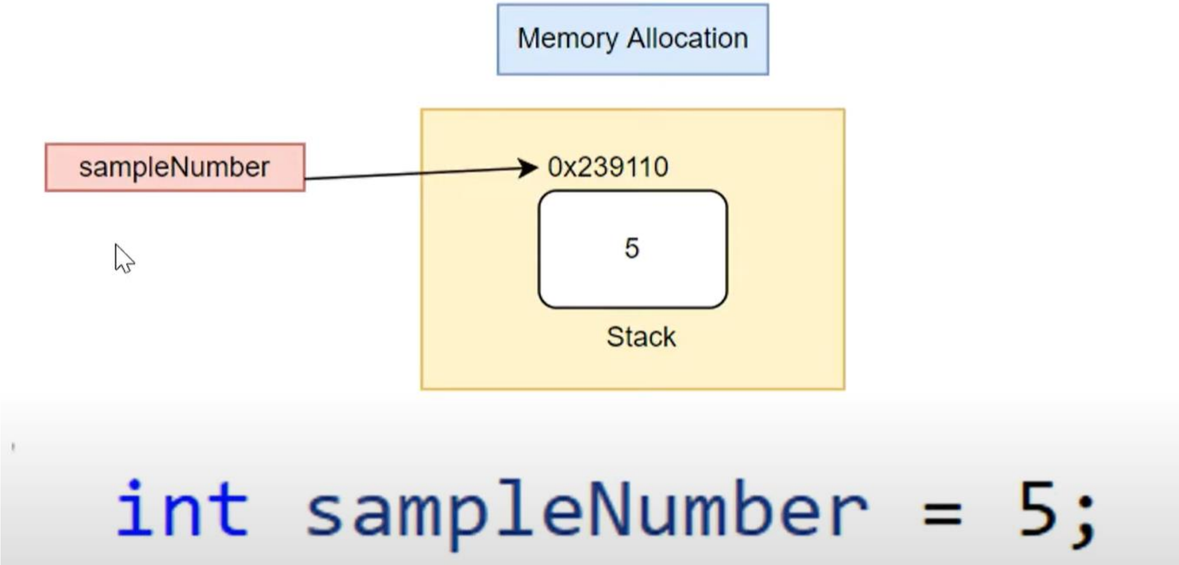
Code Example

```
int number = 10;         // Integer type
double price = 99.99;    // Floating-point type
char grade = 'A';        // Character type
bool isPassed = true;    // Boolean type
string message = "Hello!"; // String type
```

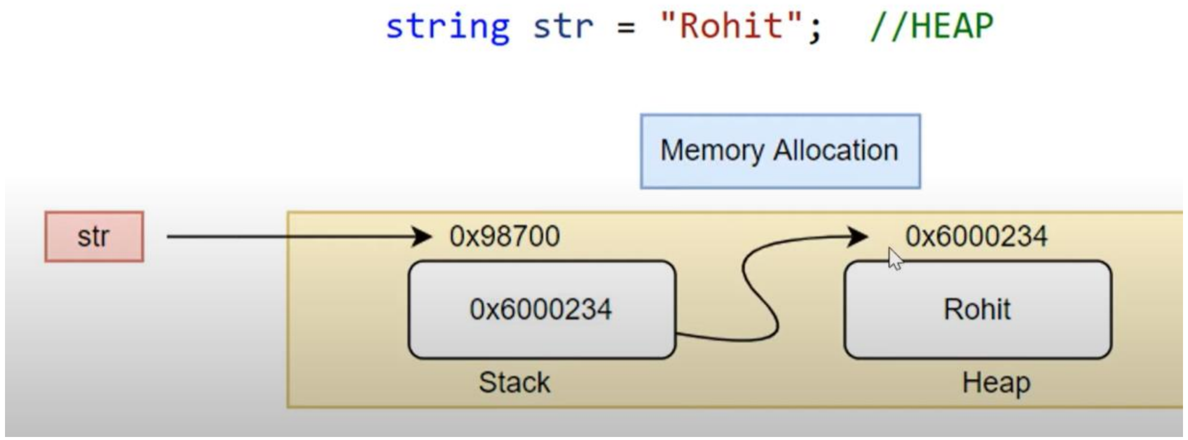
Data Types


C# Data Types
Value Data Types
Pointer Data Type
Reference Data Types
Pre Defined Data Type
User Defined Data Type
Pre Defined Data Type
User Defined Data Type
Integer
Boolean
Float
Char
Enumeration
Structure
Objects
Strings
Class
Interface

Value Type vs Reference Type

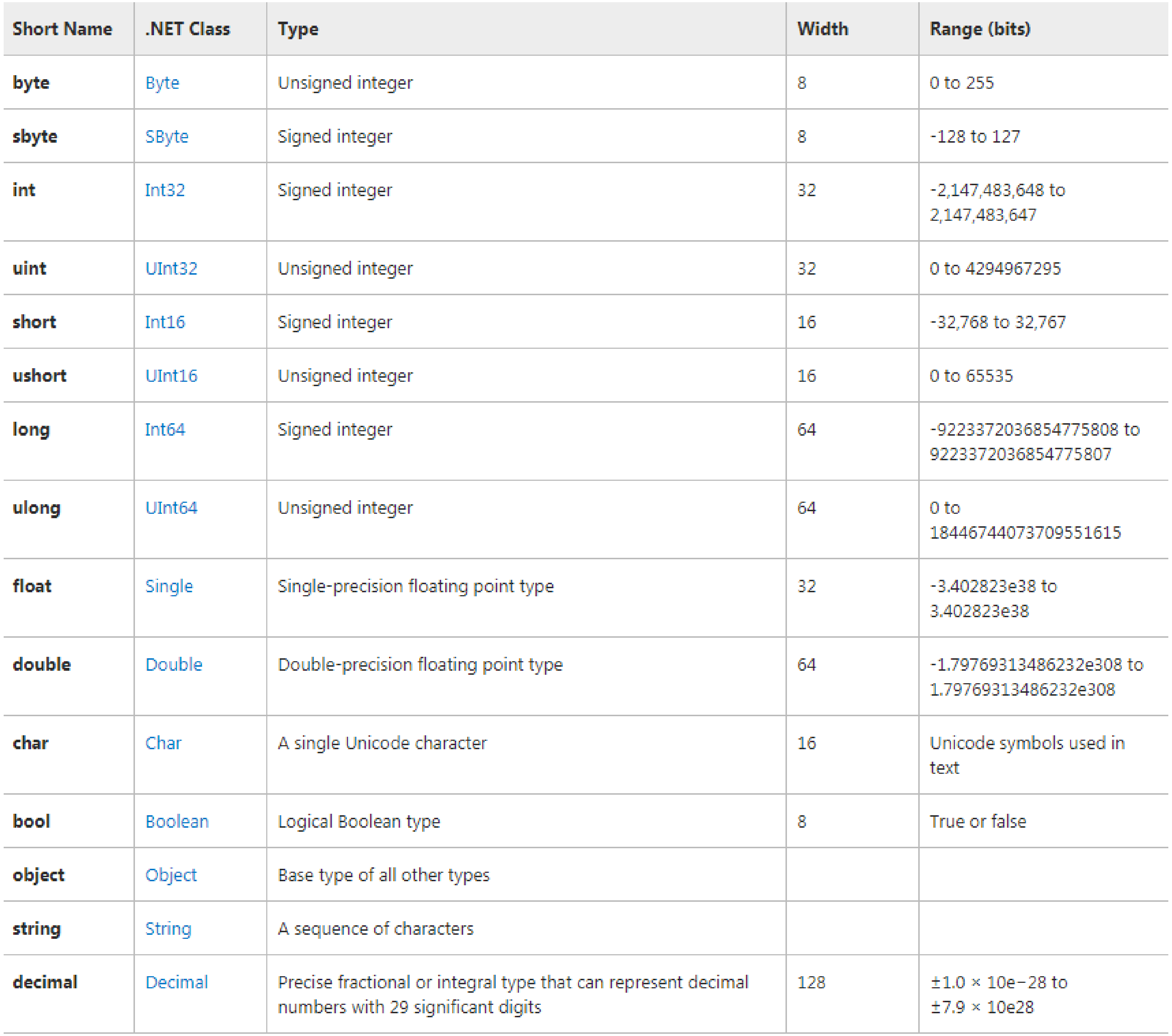


```
int sampleNumber = 5;
```



```
string str = "Rohit";  //HEAP
```

Data Types



Short Name	.NET Class	Type	Width	Range (bits)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65535
long	Int64	Signed integer	64	-9223372036854775808 to 9223372036854775807
ulong	UInt64	Unsigned integer	64	0 to 18446744073709551615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or false
object	Object	Base type of all other types		
string	String	A sequence of characters		
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	±1.0 × 10e−28 to ±7.9 × 10e28

Implicit Conversion

Implicit conversion is automatically handled by C# when a smaller data type is converted to a larger data type (e.g., int to double).

```
int marks = 85;
double gradePoint = marks / 10.0;  // Implicit conversion from int to double
```

When dividing by 10.0 (a double), C# automatically converts the integer result to a double without requiring explicit casting.

Explicit Conversion

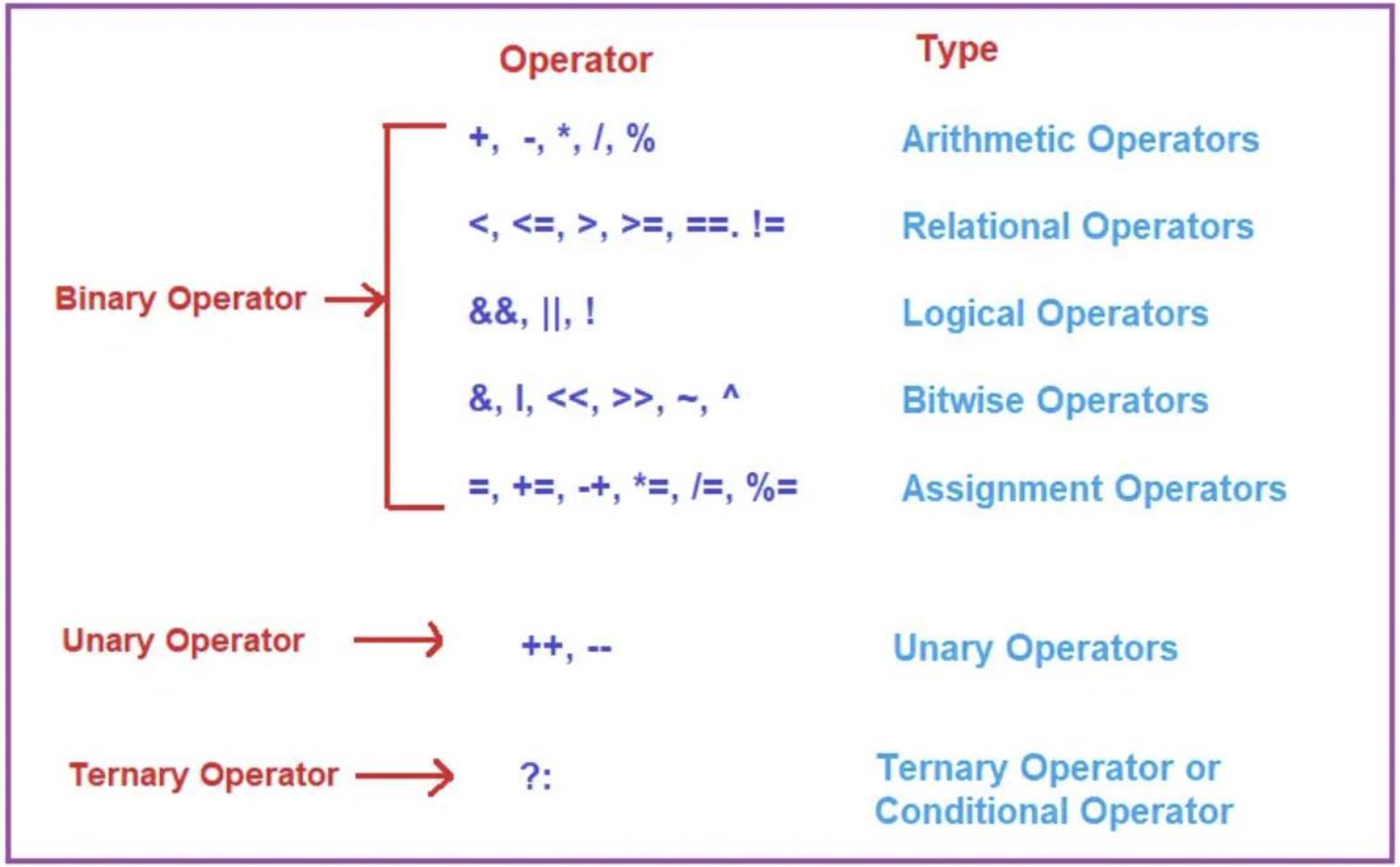
Explicit conversion is needed when converting from a larger data type to a smaller one (e.g., double to int), and it requires a cast.

```
double gradePoint = 3.6;
int truncatedGradePoint = (int)gradePoint;  // Explicit conversion from double to int
```

The decimal part of the gradePoint is truncated when explicitly converted to int.

Operators

Symbols or keywords used to perform operations on variables and values.



Operator

Type

Binary Operator

+, -, *, /, % — Arithmetic Operators

<, <=, >, >=, ==. != — Relational Operators

&&, ||, ! — Logical Operators

&, |, <<, >>, ~, ^ — Bitwise Operators

=, +=, -+, *=, /=, %= — Assignment Operators

Unary Operator ++, -- — Unary Operators

Ternary Operator ?: — Ternary Operator or Conditional Operator

Operators

Arithematic Operators

```
int a = 10;
int b = 5;
int sum = a + b;  //15
```

Relational Operator

```
bool isEqual = (a == b); // false
```

Logical

```
bool result = (a > b) && (b > 0); // true
```

Bitwise:

```
int x = 5;  // 0101 in binary
int y = 3;  // 0011 in binary
int andResult = x & y;  // 0001 (1 in decimal)
int orResult = x | y;   // 0111 (7 in decimal)
```

Operators

Assignment:

```
int num = 10;
num += 5; // num = 15
```

Unary

```
int count = 10;
count++; // Increment by 1, count = 11
count--;
```

Ternary

```
int age = 18;
string result = (age >= 18) ? "Adult" : "Minor";
```

Variable Scope

What is Variable Scope?

The area in code where a variable is accessible.

Types:

Local Variables: Declared inside methods; accessible only within those methods.

Global Variables: Declared outside methods; accessible throughout the class.

```
public class Program {
     int globalVar = 10; // Global Variable

    public void Method() {
        int localVar = 5; // Local Variable
      }
    }
```

Best Practices

- Use meaningful variable names.
- Declare variables with the smallest scope possible.
- Initialize variables before using them.
- Choose the appropriate data type for the variable.

C# Exercise

Write a C# program that asks the user to input two integer numbers. Perform the following operations and display the results:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Increment the first number (++) and display the result.
- Decrement the second number (--) and display the result.

Example Input/Output:

Input:

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

Output:

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3
Modulus: 1
After Increment, First Number: 11
After Decrement, Second Number: 2
```

C# Exercise

Write a C# program to compare two numbers entered by the user. Perform the following:

- Use relational operators (>, <, >=, <=, ==, !=) to compare the two numbers and display the results.
- Use logical operators (&&, ||, !) to check the following conditions:
  - Both numbers are greater than 10.
  - At least one of the numbers is even.
  - Neither number is negative.

Input:

```
Enter the first number: 15
Enter the second number: 8
```

Output:

```
First number > Second number: True
First number < Second number: False
First number >= Second number: True
First number <= Second number: False
First number == Second number: False
First number != Second number: True
Both numbers are greater than 10: False
At least one number is even: True
Neither number is negative: True
```

Condition Statements

Condition statements help control the flow of the program based on whether a condition is true or false.

They are crucial for decision-making in programming.

If-Else, Switch, and Ternary Operators

If-Else Statement

- The if statement checks whether a condition is true.
- The else block runs when the condition is false.

Syntax

```
if (condition) {
// code block if condition is true
}
else {
// code block if condition is false
}
```

Code :

```
int number = 10;
if (number > 5)
{
Console.WriteLine("Number is greater than 5");
 }
else
{
Console.WriteLine("Number is not greater than 5"); }
```

Else-If Ladder

- An else if ladder allows multiple conditions to be tested sequentially.
- If the first condition fails, the next condition is checked, and so on.

Syntax

```
if (condition1) {
// code block if condition is true
}
else if (condition2)
{
 // code block if condition2 is true
}
else {
// code block if condition is false
}
```

Code :

```
int number = 15;
if (number > 20)
{
    Console.WriteLine("Number is greater than 20");
}
else if (number > 10)
{
    Console.WriteLine("Number is greater than 10 but less than or equal to 20");
}
else
{
    Console.WriteLine("Number is 10 or less");
}
```

Switch Statement

- The switch statement is used when there are multiple possible values for a variable.
- It is often more readable than multiple if-else statements.

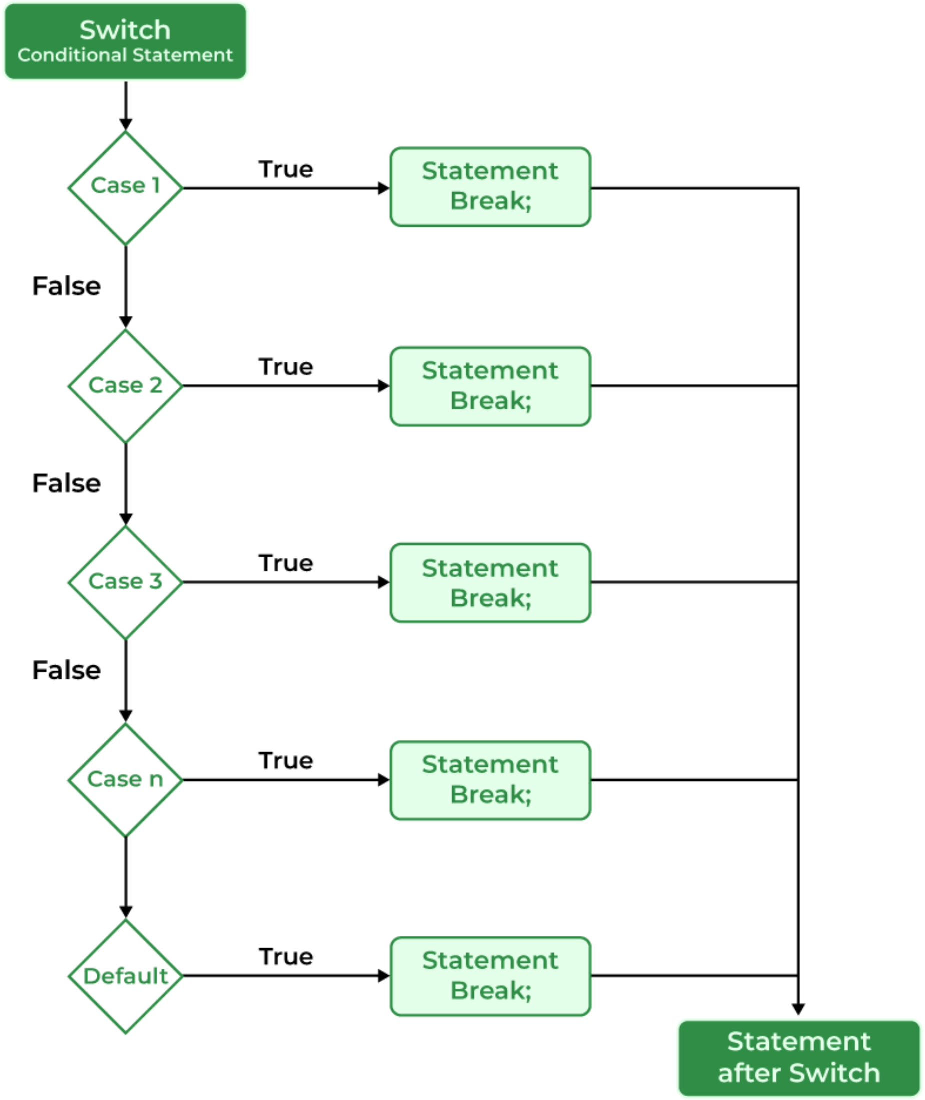
Syntax

```
switch (variable)
{
    case value1:
        // code block if variable == value1
        break;
    case value2:
        // code block if variable == value2
        break;
    default:
        // code block if no case matches
        break;
}
```

Switch Statement

Code

```
int day = 3;
switch (day)
{
    case 1:
        Console.WriteLine("Monday");
        break;
    case 2:
        Console.WriteLine("Tuesday");
        break;
    case 3:
        Console.WriteLine("Wednesday");
        break;
    default:
        Console.WriteLine("Invalid day");
        break;
}
```



Ternary Operator

- The ternary operator is a shorthand for `if-else` statements.
- It is useful for assigning values based on a condition.

Syntax

condition ? expression_if_true : expression_if_false;

Code :

```
int age = 20;
string result = age >= 18 ? "Adult" : "Minor";
Console.WriteLine(result);
```

Nested Condition Statements

Condition statements can be nested inside one another to handle more complex conditions.

Code :

```
int age = 25;
bool hasTicket = true;

if (age >= 18)
{
    if (hasTicket)
    {
        Console.WriteLine("You can enter the event.");
    }
    else
    {
        Console.WriteLine("You need a ticket to enter.");
    }
}
else
{
    Console.WriteLine("You must be 18 or older to enter.");
}
```

C# Exercise

Write a C# program that asks the user to enter their **marks** (an integer between 0 and 100). Based on the marks, the program should assign a grade and grade point according to the Nepali grading system:

- **Above 90**: Grade **A+**, Grade Point **4.0**, Comment **Outstanding**.
- **80 to 90**: Grade **A**, Grade Point **3.6**, Comment **Excellent**.
- **70 to 80**: Grade **B+**, Grade Point **3.2**, Comment **Very Good**.
- **60 to 70**: Grade **B**, Grade Point **2.8**, Comment **Good**.
- **50 to 60**: Grade **C+**, Grade Point **2.4**, Comment **Satisfactory**.
- **40 to 50**: Grade **C**, Grade Point **2.0**, Comment **Acceptable**.
- **30 to 40**: Grade **D+**, Grade Point **1.6**, Comment **Partially Acceptable**.
- **20 to 30**: Grade **D**, Grade Point **1.2**, Comment **Insufficient**.
- **0 to 20**: Grade **E**, Grade Point **0.8**, Comment **Very Insufficient**.

If the marks are outside the valid range (less than 0 or greater than 100), print:

"Invalid input! Please enter marks between 0 and 100."

Input:

```
Enter your marks: 92
```

Output:

```
Grade: A+
Grade Point: 4.0
Comment: Outstanding
```