Chapter 1:

1.1 Introduction to C# - its features and applications:

C# (pronounced "C sharp") is a powerful, object-oriented programming language developed by Microsoft. It combines the features of C and C++ with the simplicity and ease of use of Visual Basic. Some key features of C# include:

- Object-Oriented: C# supports encapsulation, inheritance, and polymorphism, allowing developers to create reusable and modular code.
- Type Safety: C# enforces strong type checking at compile-time, reducing the likelihood of runtime errors.
- Garbage Collection: C# automatically manages memory by reclaiming objects that are no longer in use, reducing the burden on developers.
- Integration with .NET Framework: C# is tightly integrated with the .NET Framework, providing access to a vast library of pre-built classes and components.
- Cross-Platform Development: With the introduction of .NET Core, C# can be used to build applications that run on various platforms, including Windows, macOS, and Linux.

Applications of C# include:

- **Desktop Applications:** C# can be used with frameworks like Windows Presentation Foundation (WPF) and Windows Forms to create intuitive and feature-rich desktop applications.
- **Web Applications:** C# can be used with ASP.NET and ASP.NET Core to develop robust and scalable web applications.
- **Mobile Applications:** C# can be used with Xamarin or .NET MAUI to build cross-platform mobile apps for iOS and Android.
- **Game Development:** C# is widely used in game development with frameworks like Unity, providing a high-level language for scripting game logic and behavior.
- **IoT and Embedded Systems:** C# can be used with frameworks like .NET IoT to build applications for Internet of Things (IoT) devices and embedded systems.

1.2 Structure of C#:

The structure of a C# program consists of various elements organized in a specific manner. A basic structure of a C# program includes:

1. **Namespace Declaration:** It defines the namespace in which the program resides, helping to organize and group related code.

```
namespace MyNamespace
{
   // Program code goes here
}
```

2. **Class Declaration:** It defines a class, which is a blueprint for creating objects. Classes encapsulate data and behavior into a single unit.

```
namespace MyNamespace
{
   class MyClass
   {
      // Class members go here
   }
}
```

3. **Main Method:** It serves as the entry point for the program and contains the code that will be executed first.

```
namespace MyNamespace
{
   class MyClass
   {
      static void Main(string[] args)
      {
         // Code to be executed goes here
      }
   }
}
```

1.3 Variables in C#:

Variables in C# are used to store and manipulate data. They have a data type, a name, and a value. C# supports various built-in data types, including integers, floating-point numbers, characters, booleans, and strings. Here are some examples of variable declarations and assignments:

```
int age; // Declaration of an integer variable
age = 25; // Assignment of a value to the age variable

double height = 1.75; // Declaration and assignment of a floating-point variable

char grade = 'A'; // Declaration and assignment of a character variable

bool isStudent = true; // Declaration and assignment of a boolean variable

string name = "John"; // Declaration and assignment of a string variable
```

1.4 Identifiers in C#:

Identifiers in C# are used to give names to various program elements such as variables, classes, methods, and namespaces. They must follow specific rules and conventions. Here are some guidelines for naming identifiers in C#:

- They must start with a letter or an underscore.
- They can contain letters, digits, and underscores.
- They are case-sensitive, meaning "myVariable" and "myvariable" are considered different.
- They cannot be a reserved keyword.
- It is recommended to use descriptive and meaningful names.

Examples of valid identifiers:

```
int age;
string firstName;
double _salary;
```

1.5 Keywords in C#:

Keywords in C# are reserved words that have a special meaning in the language and cannot be used as identifiers. They represent language constructs and control the behavior of the program. Examples of C# keywords include if, else, switch, for, while, class, int, string, and many others.

```
int age = 25;
if (age > 18)
{
   Console.WriteLine("You are an adult.");
}
else{
 Console.WriteLine("You are a kid.");

}
```

In the example above, if and else are keywords used for conditional statements, while int is a keyword used for declaring an integer variable.

1.6 Data Types in C#:

Data types in C# specify the type of data that can be stored in variables. C# supports two categories of data types: value types and reference types.

- **Value types** represent simple data types, such as integers, floating-point numbers, booleans, and characters. They are stored directly in memory and occupy a fixed size.

Examples of value types:

```
int age = 25;
double height = 1.75;
bool isStudent = true;
char grade = 'A';
```

- **Reference types** represent complex data types, such as classes, interfaces, arrays, and delegates. They store a reference (memory address) to the actual data, which is allocated on the heap. Examples of reference types:

```
string name = "John";
object obj = new object();
int[] numbers = new int[] { 1, 2, 3 };
```

1.7 C# Type Conversion:

Type conversion, also known as type casting, is the process of converting a value from one data type to another. C# supports two types of type conversions: implicit and explicit.

- **Implicit conversion** is done automatically by the compiler when there is no risk of data loss or precision. For example:

```
int number = 10;
double result = number; // Implicit conversion from int to double
```

- **Explicit conversion** requires the use of casting operators and may result in data loss or exceptions. For example:

```
double value = 3.14;
int approximation = (int)value; // Explicit conversion from double to int
```

1.8 C# Operators:

Operators in C# are symbols or keywords that perform specific operations on operands (variables or values). C# supports a wide range of operators for arithmetic, assignment, comparison, logical operations, and more.

1. **Arithmetic operators** are used for performing mathematical calculations:

```
int a = 5;
int b = 2;
int sum = a + b; // Addition
int difference = a - b; // Subtraction
int product = a * b; // Multiplication
int quotient = a / b; // Division
int remainder = a % b; // Modulus (remainder)
```

2. **Assignment operators** are used to assign values to variables:

```
int x = 10;
x += 5; // Equivalent to x = x + 5
x -= 3; // Equivalent to x = x - 3
```

3. **Comparison operators** are used to compare values:

```
int a = 5;
int b = 3;
bool isEqual = (a == b); // Equality comparison
bool isGreaterThan = (a > b); // Greater than comparison
bool isLessThan = (a < b); // Less than comparison
```

4. **Logical operators** are used for logical operations:

```
bool condition1 = true;
bool condition2 = false;
bool result = (condition1 && condition2); // Logical AND
bool result2 = (condition1 || condition2); // Logical OR
bool result3 = !condition1; // Logical NOT
```

These are just a few examples of C# operators. Understanding and using operators effectively is crucial for performing various calculations, comparisons, and logical operations in C#.