.NET

UNIT I - Introduction to .NET

SOFTWARE DEVELOPMENT TOOLS(C# and ASP.NET)

II MSc CS

V.B.Buvaneswari

CONTENTS OF THIS UNIT

Introduction

- Architecture of .Net Framework
- Common Language Runtime
- Common type system
- Class libraries
- Microsoft Intermediate Languages
- JItters
- Unmanaged code

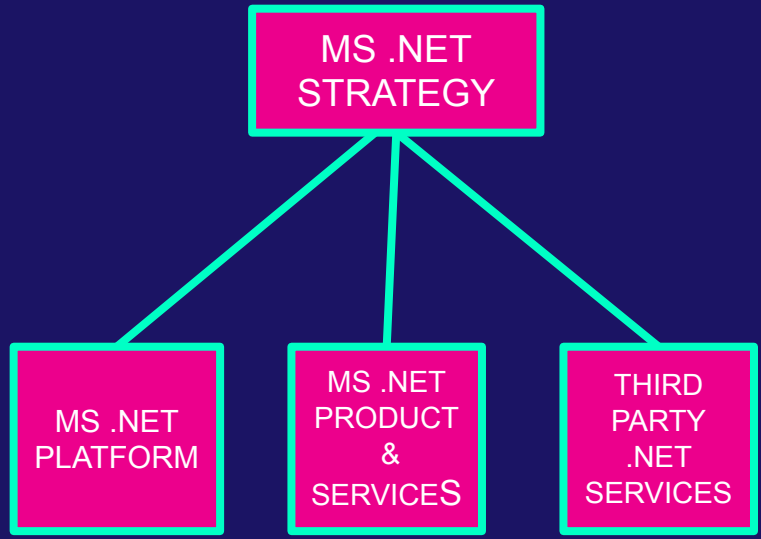
Introduction to C#

- Evolution of C#
- Characteristics of C#
- How does C# differ from C++ and Java
- Data types
- Variables and Literals
- Boxing and unboxing
- Operators and Expressions
- Type conversions
- Mathematical functions
- Decision making and branching
- Decision making and looping

INTRODUCTION To .NET

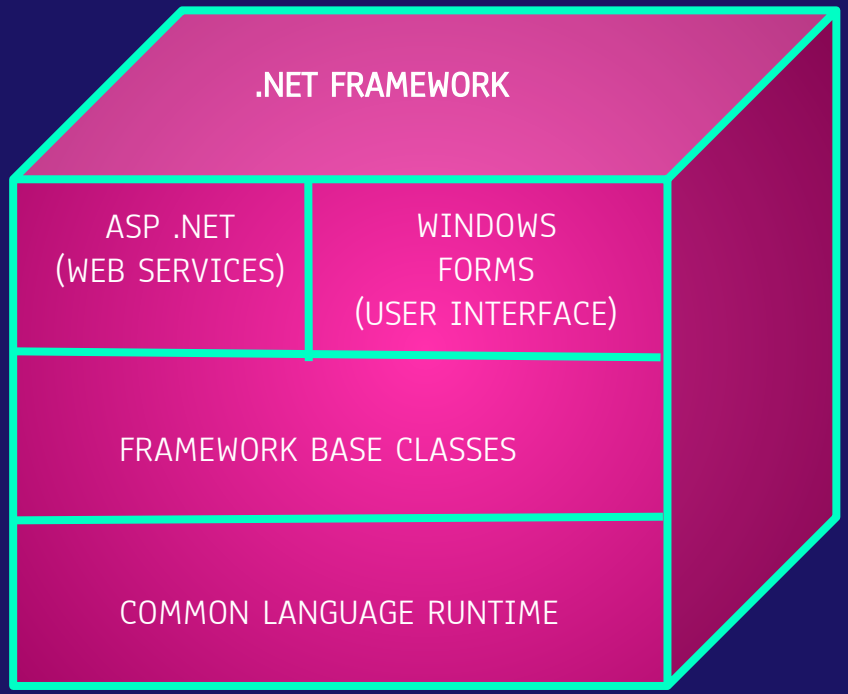
INTRODUCTION TO .NET

- A software framework that includes everything required for developing software for web services.
- Integrates presentation technologies, component technologies and data technologies on a single platform so as to enable users to develop Internet applications as easily as they do on desktop systems.
- Microsoft took many of the best ideas in the industry, added their own creativity and innovations and produced a coherent systems solution popularly known as Microsoft .NET.
- The Microsoft .NET software solution strategy includes three key approaches shown in the figure.



ARCHITECTURE OF .NET FRAMEWORK

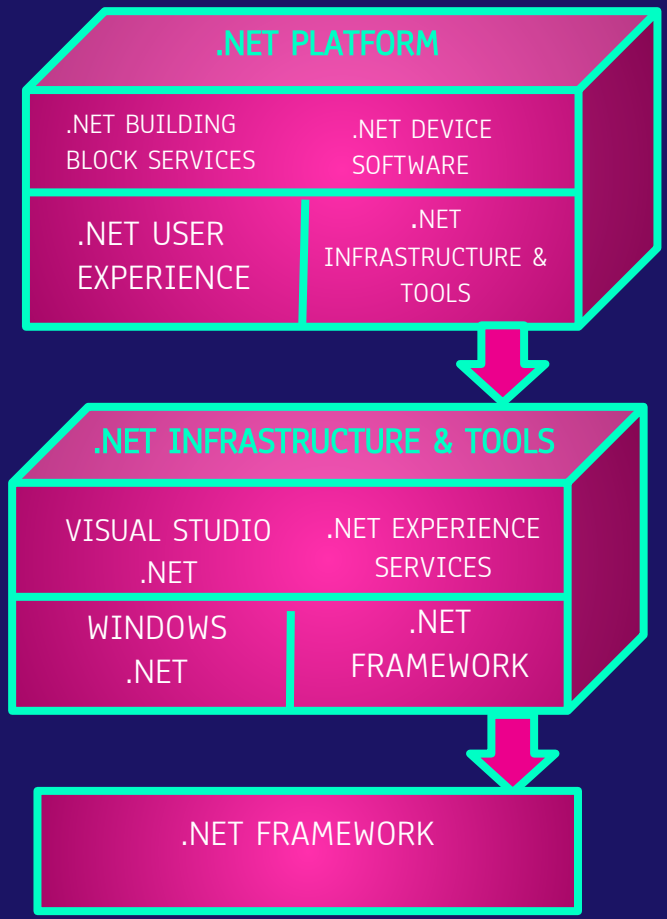
- Provides an environment for building, deploying and running web services and other applications. Consists of three distinct technologies as shown in Figure
- Common Language Runtime (CLR)
- Framework Base Classes
- User and program interfaces(ASP .NET and Winforms)



ARCHITECTURE OF .NET FRAMEWORK

- The CLR is the core of the .NET Framework and is responsible for loading and running C# programs.
- Base classes provide basic data types, collection classes and other general classes for use by C# and other .NET languages.
- The top layer contains a set of classes for developing web services and to deal with the user interface.

COMPONENTS OF .NET FRAMEWORK

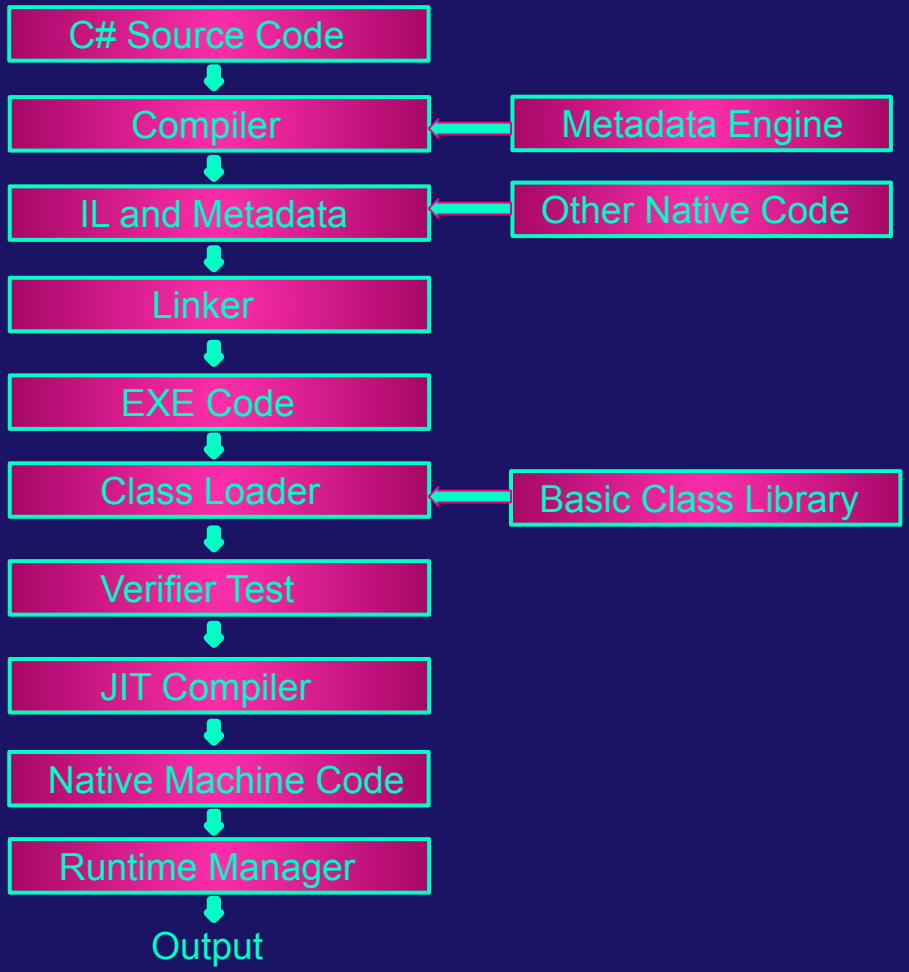


COMMON LANGUAGE RUNTIME

- The Common Language Runtime, popularly known as CLR is the heart and soul of the .NET Framework.
- CLR is a runtime environment in which programs written in C# and other .
- NET languages are executed.
- Supports cross-language interoperability.
- The CLR provides a number of services that include:
  - 😊 Loading and execution of programs
  - 😊 Memory isolation for applications
  - 😊 Verification of type safety
  - 😊 Compilation of JL into native executable code
  - 😊 Providing metadata
  - 😊 Memory management (automatic garbage collection)
  - 😊 Enforcement of security
  - 😊 Interoperability with other systems
  - 😊 Managing exceptions and errors
  - 😊 Support for tasks such as debugging and profiling

COMPONENTS OF CLR

CLR ACTIVITIES FOR EXECUTING A PROGRAM



CLR ACTIVITIES FOR EXECUTING A PROGRAM (CONT.)

- The flow chart shows CLR activities that go on when an application is executed.
- The source code is compiled to IL while the metadata engine creates metadata information.
- IL and metadata are linked with other native code if required and the resultant IL code is saved.
- During execution, the IL code and any requirement from the base class library are brought together by the class loader.
- The combined code is tested for type-safety and then compiled by the JIT compiler to produce native machine code, which is sent to the runtime manager for execution.

COMMON TYPE SYSTEM(CTS)

- The .NET Framework provides multiple language support using the feature known as Common Type System that is built into the CLR.
- The CTS supports a variety of types and operations found in most programming languages and therefore calling one language from another does not require type conversions.
- Although C# is specially designed for the .NET platform, .NET programs can be built in a number of other languages including C++ and Visual Basic.

COMMON LANGUAGE SPECIFICATION(CLS)

- The CLS defines a set of rules that enables interoperability on the .NET platform.
- These rules serve as a guide to third-party compiler designers and library builders.
- The CLS is a subset of CTS and therefore the languages supporting the CLS can use each others' class libraries as if they are their own.
- Application Program Interfaces (API's) that are designed following the rules of CLS can easily be used by all the .NET languages.

CLASS LIBRARIES

- NET supplies a library of base classes that can be used to implement applications quickly.
- Can be used by simply instantiating them and invoking their methods or by inheriting them through derived classes, thus extending their functionality.
- Much of the functionality in the base framework classes resides in the vast name space called System.

CLASS LIBRARIES (cont.)

The base classes in the system namespace for many different tasks including:

- Input/Output Operations
- String handling
- Managing arrays, lists, maps, etc
- Accessing files and file systems
- Accessing the registry
- Security
- Windowing
- Windows messages
- Database management
- Evaluation of mathematical functions
- Drawing
- Managing errors and exceptions
- Connecting to the Internet

MICROSOFT INTERMEDIATE LANGUAGE (MIL)

- MSIL, or simply IL, is an instruction set into which all the .NET programs are compiled.
- lt is akin to assembly language and contains instructions for loading, storing, initializing and calling methods.
- A C# program or any program written in a CLS-compliant language, the source code is compiled into MSIL.

JITters

Three different JITters can be used to convert the MSIL into native code, depending on the circumstances: -

1. Install-time code generation
   - Will compile an entire assembly into CPU-specific binary code.
   - An assembly is the code package that's sent to the compiler.
   - This compilation is done at install time
   - The advantage of install-time code generation is that it allows user to compile the entire assembly just once before you run it.
   - Because the entire assembly is compiled, user don't have to worry about intermittent performance issues every time a method in your code is executed the first time.
   - Usage of this utility depends on the size of specific system and deployment environment.

JITters (cont.)

2. JIT
   - The default JITter is called at run time-in a method is invoked for the first time.
   - This is akin to a "pay-as-you-go" plan and is the default if user don't explicitly run the PreJIT compiler.

JITters (cont.)

3. EconoJIT
   - Another run-time JITter specifically designed for systems that have limited resources-for example, handheld devices with small amounts of memory.
   - The major difference between this JITter and the regular JITter is the incorporation of something called *code pitching*.
   - Code pitching allows the EconoJIT to discard the generated, or compiled, code if the system begins to run out of memory.
   - The benefit is that the memory is reclaimed. However, the disadvantage is that if the code being pitched is invoked again, it must be compiled again as though it had never been called.

MANAGED CODE

- The CLR is responsible for managing the execution of code compiled for the .NET platform.
- The code that satisfies the CLR at runtime in order to execute is referred to as managed code. Compilers that are compatible to the .NET platform generate managed code.
- For example, the C# compiler generates managed code.
- The managed code generated by C# (and other compilers capable of generating managed code) is IL code.
- The IL code is then converted to native machine code by the JIT compiler

UNMANAGED CODE

- Code that executes under the control of the runtime is called managed code.
- Conversely, code that runs outside the runtime is called unmanaged code.
- COM components, ActiveX interfaces, and Windows API functions are examples of unmanaged code.

INTRODUCTION To C#

EVOLUTION OF C#

- The World Wide Web is growing everyday not only in terms of the number of web sites but also in terms of volume and variety of information it contains.
- Doing business across those web sites is very exciting and useful.
- The number of individuals and organizations using the Internet, which supports the World Wide Web is increasing exponentially.
- The Internet is therefore in the mainstream of many business organizations today.

EVOLUTION OF C#

- There are number of limitations in using the WWW over the Internet.
- User can see only one site at a time.
- The site has to be authored to our hardware environment
- The information user get is basically read-only
- Dynamically similar information stored in different sites cannot be compared.
- The Internet is a collection of many information islands that do not cooperate with each other.
- It continues to be a browsing and presentation network rather than an intelligent knowledge management network.

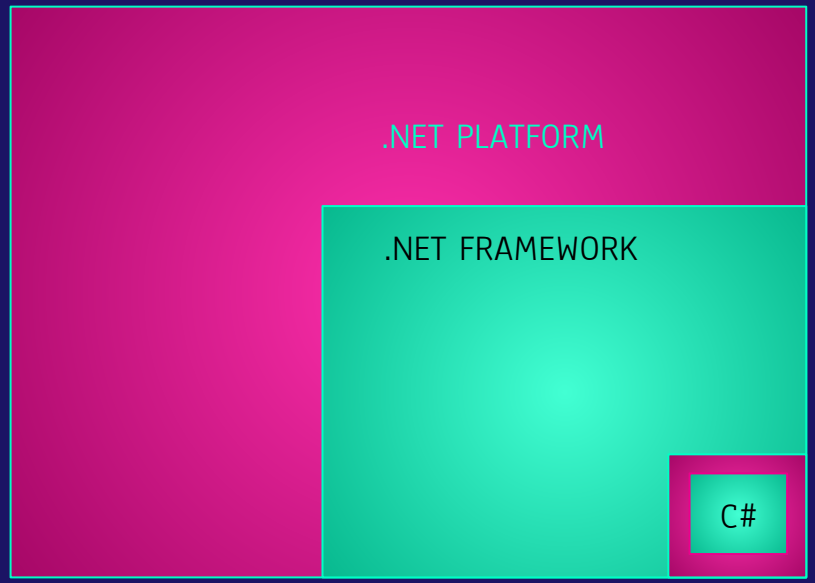
EVOLUTION OF C#

- Microsoft Chairman Bill Gates, the architect of many innovative and path-breaking software products during the past two decades, wanted to develop a software platform which will overcome these limitations and enable users to get information anytime and anywhere, using a natural interface.
- The platform should be a collection of readily available Web services that can be distributed and accessed via standard Internet protocols.
- The Web has to be both programmable and intelligent.
- The outcome is a new generation platform called .NET.
- .NET is simply the Microsoft's vision of 'software as a service.'

EVOLUTION OF C#

- Although the research and development work of .NET platform began in the mid-90s, only during the Microsoft Professional Developers Conference in September 2000, was .NET officially announced to the developer community.
- At the same conference, Microsoft introduced C# as a de facto language of the .NET platform.
- C# had already been used to code key modules of the .NET platform.
- C# has been particularly designed to build software components for .NET ~ and it supports key features of .NET natively.
- They are fairly tightly tied together.
- C# compiler is embedded into .

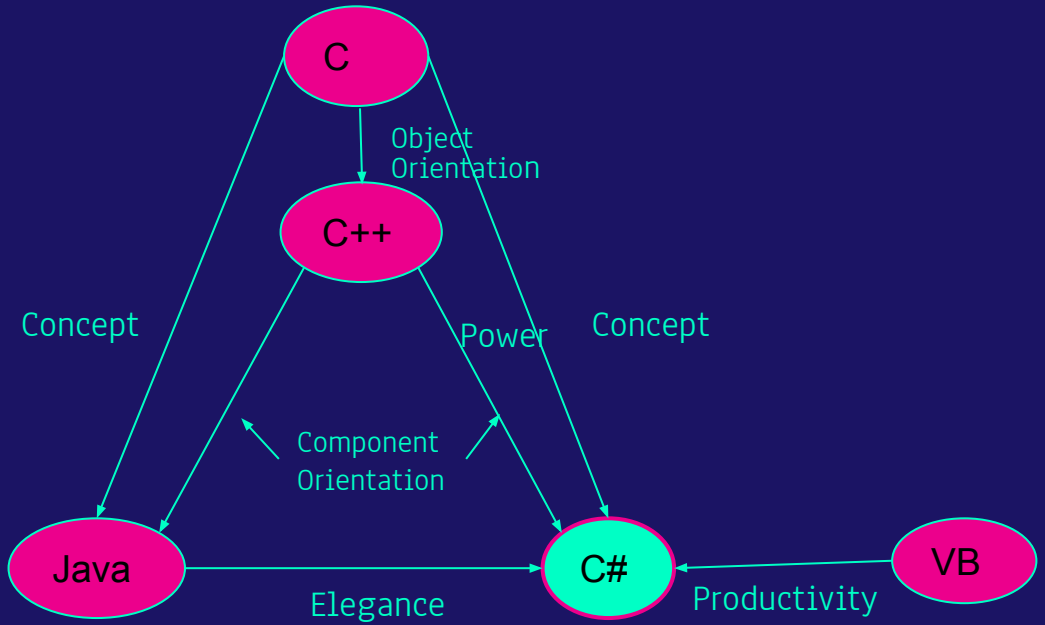
C# INSIDE THE .NET



EVOLUTION OF C#

- Like Java, C# is a descendant of C++ which in turn is a descendant of C .
- C is the mother of all the three modern languages.
- C# modernizes C++ by enhancing some of its features and adding a few new features so as to help developers do more with fewer lines of code and fewer opportunities for error.
- C# borrows Java's features such as grouping of classes, interfaces and implementation together in one file so that programmers can edit the code more easily.
- C# also handles objects using references, the same way as Java.
- C# uses VB's approach to form design, namely, dragging controls from a tool box, dropping them onto forms, and writing event handlers for them.
- C# applies this process not only to the development of user interface but also to the building of business objects.

EVOLUTION OF C# LANGUAGE



CHARACTERISTICS OF C#

1.Simple

- C# simplifies C++ by eliminating irksome operators such as ->, :: and pointers.
- C# treats integer and Boolean data types as two entirely different types.
- This means that the use of= in place of == in if statements will be caught by the compiler.

2.Consistent

- C# supports an unified type system which eliminates the problem of varying ranges of integer types.
- All types are treated as objects and developers can extend the type system simply and easily.

CHARACTERISTICS OF C#

3.Modern

- C# is called a modern language due to a number of features it supports.
  - 😊 Automatic garbage collection
  - 😊 Rich intrinsic model for error handling
  - 😊 Decimal data type for financial applications
  - 😊 Modem approach to debugging
  - 😊 Robust security model

4.Object-oriented

- C# is truly object-oriented and supports all the three tenets of object-oriented systems
  - 😊 Encapsulation
  - 😊 Inheritance
  - 😊 Polymorphism
- The entire C# class model is built on top of the Virtual Object System (VOS) of the .NET Framework
- In C#, everything is an object and there are no more global functions, variables and constants.

CHARACTERISTICS OF C#

5.Versionable

- Making new versions of software modules work with the existing applications is known as versioning.
- C# provides support for versioning with the help of new and override keywords.
- With this support, a programmer can guarantee that his new class library will maintain binary compatibility with the existing client applications.

6.Compatible

- C# enforces the .NET common language specifications and therefore allows inter-operation with other .NET languages.
- C# provides support for transparent access to standard COM and OLE Automation.
- C# also permits interoperation with C-style APis.

CHARACTERISTICS OF C#

7.Flexible

- Although C# does not support pointers, User may declare certain classes and methods as 'unsafe' and then use pointers to manipulate them.
- However, these codes will not be type-safe.

8.Inter-operability

- C# provides support for using COM objects, no matter what language was used to author them.
- C# also supports a special feature that enables a program to call out any native API.

CHARACTERISTICS OF C#

9.Type-safe

Type-safety promotes robust programs. C# incorporates a number of type-safe measures.

- All dynamically allocated objects and arrays are initialized to zero
- Use of any uninitialized variables produces an error message by tile compiler
- Access to arrays are range-checked and warned ifit goes out-of-bounds
- C# does not permit unsafe casts
- C# enforces overflow checking in arithmetic operations
- Reference parameters that are passed are type-safe
- C# supports automatic garbage collection

HOW DOES C# DIFFER FROM C++?

Change 1

C# compiles straight from source code to executable code, with no object files.

Change 2

C# does not separate class definition from implementation. Classes are defined and implemented in the same place and therefore there is no need for header files.

Change 3

ln C#, class definition does not use a semicolon at the end.

Change 4

The first character of the Main( ) function is capitalized. The Main must return either int or void type value.

Change 5

C# does not support #include statement.

Change 6

All data types in C# are inherited from the object super class and therefore they are objects

HOW DOES C# DIFFER FROM C++?

Change 7

All the basic value types will have the same size on any system. This is not the case in C or C++. Thus C# is more suitable for writing distributed applications.

Change 8

In C#, data types belong to either value types (which are created in a stack) or reference types (which are created in a heap).

Change 9

C# checks for uninitialized variables and gives error messages at compile time. In C++, an uninitialized variable goes undetected thus resulting in unpredictable output.

Change 10

In C#, structs are value types.

Change 11

C# supports a native string type. Manipulation of strings are easy.

Change 12

C# supports a native Boolean data type and bool type data cannot be implicitly or explicitly cast to any data type except object.

HOW DOES C# DIFFER FROM C++?

Change 13

C# declares null as a keyword and considers it as an intrinsic value

Change 14

C# does not support pointer types for manipulating data. However, they are used in what is known as 'unsafe' code.

Change 15

Variable scope rules in C# are more restrictive. ln C#, duplicating the same name within a routine is illegal, even if it is in a separate code block.

Change 16

C# permits declaration of variables between goto and label.

Change 17

Objects in C# can only be created using the new keyword.

Change 18

Arrays are classes in C# and therefore they have built-in functionality for operations such as sorting, searching and reversing.

HOW DOES C# DIFFER FROM C++?

Change 19

Arrays in C# are declared differently and behave very differently compared to C++ arrays.

Change 20

C# provides special syntax to initialize arrays efficiently.

Change 21

Arrays in C# are always reference types rather than value types, as they are in C++ and therefore stored in a heap.

Change 22

C#, expressions in if and while statements must resolve to a bool value. Accidental use of the assignment operator(=) instead of equality operator == will be caught by the compiler

Change 23

C# supports four iteration statements rather than three in C++. The fourth one is the foreach statement.

Change 24

C# does not allow silent fall-through in switch statements. lt requires an explicit jump statement at the end of each case statement.

HOW DOES C# DIFFER FROM C++?

Change 25

In C#, switch can also be used on string values

Change 26

C# does not support the labeled break and therefore jumping out of nested loops can be messy.

Change 27

The set of operators that can be overloaded in C# is smaller compared to C++.

Change 28

C# can check overflow of arithmetic operations and conversions using checked and unchecked keywords.

Change 29

C# does not support default arguments

Change 30

Variable method parameters are handled differently in C#

HOW DOES C# DIFFER FROM C++?

Change 31

In exception-handling, unlike in C++, any type cannot be thrown in C#. The thrown value has to be a reference to a derived class or **System.Exception** object.

Change 32

C# requires ordering of catch blocks correctly.

Change 33

General catch statement catch (...) in C++ is replaced by simple catch in

Change 34

C# does not provide any defaults for constructors.

Change 35

Destructors in C# behave differently than in C++

Change 36

In C#, static members cannot be accessed via an object, as in C++

HOW DOES C# DIFFER FROM C++?

Change 37

C# does not support multiple code inheritance

Change 38

Casting in C# is much safer than in C++

Change 39

When overriding a virtual method, the override keyword must be used.

Change 40

Abstract methods in C# are similar to virtual functions in C++, but C# abstract methods cannot have implementations.

Change 41

Command line parameters array behave differently in C# as compared to C++

HOW DOES C# DIFFER FROM JAVA?

Change 1

Arrays are declared differently in C#.

Change 2

C# has more primitive data type.

Change 3

Unlike Java, all C# data types are objects.

Change 4

Although C# uses .NET runtime that is similar to Java runtime, the C# compiler produces an executable code.

Change 5

Although C# classes are quite similar to Java classes, there are a few important differences relating to constants, base classes and constructors, static constructors, versioning, accessibility of members etc.

Change 6

Java uses static final to declare a class constant while C# uses const.

HOW DOES C# DIFFER FROM JAVA?

Change 7

The new modifier used for class members has no complement in Java

Change 8

C# supports the struct type and Java does not.

Change 9

Java does not provide operator overloading.

Change 10

The convention for Java is to put one public class in each file and some compilers require this. C# allows any source file arrangement.

Change 11

In Java, class members are virtual by default and a method having the same name in a derived class overrides the base class member. In C#, the base member is required to have the virtual keyword and the derived member is required to use the override keyword.

Change 12

C# provides better versioning support than Java.

HOW DOES C# DIFFER FROM JAVA?

Change 13

C# provides static constructors for initialization

Change 14

C# includes native support for properties, Java does not.

Change 15

Java does not directly support enumeration.

Change 16

C# provides built-in delegates and events. Java uses interfaces and inner classes to achieve a similar result.

Change 17

In Java, parameters are always passed by value. C# allows parameters to be passed by reference by using the ref keyword.

Change 18

C# adds internal, a new accessibility modifier. Members with internal accessibility can be accessed from other classes within the same project, but not from outside the project.

HOW DOES C# DIFFER FROM JAVA?

Change 19

Java does not have any equivalent to C# indexers.

Change 20

C# does not allow free fall_through from case to case.

Change 21

Catch blocks should be ordered correctly in C#.

Change 22

Both Java and C# support interfaces. But, C# does not allow type definitions in interfaces, while Java interfaces can have const type data.

Change 23

ln Java, the switch statement can have only integer expression, while C# supports either an integer or string expressions.

Change 24

C# provides a fourth type of iteration statement, foreach for quick and easy iterations over collections and array type data.

HOW DOES C# DIFFER FROM JAVA?

Change 25

C# checks overflows using checked statements.

Change 26

C# uses is operator instead of instanceof operator in Java.

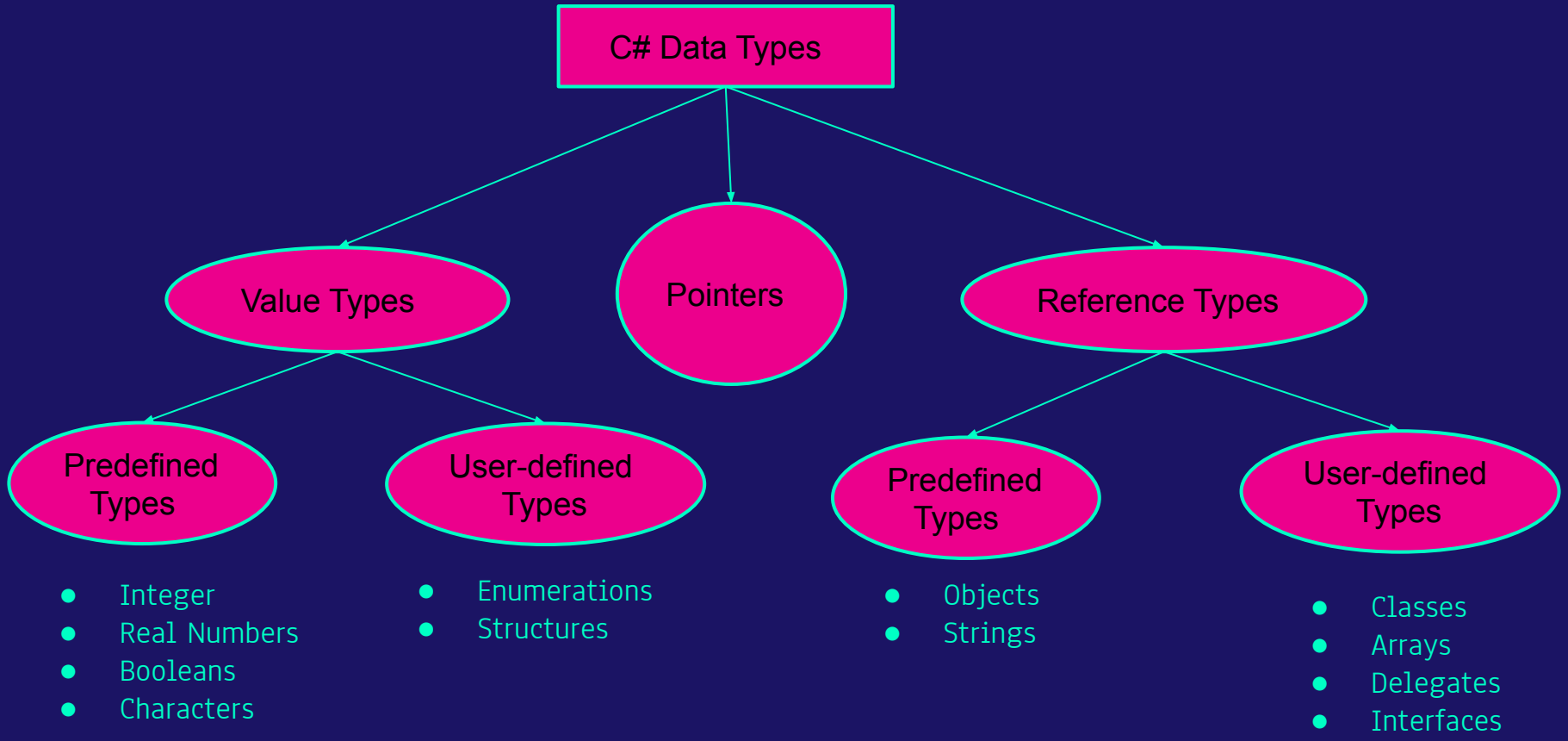
Change 27

C# allows a variable number of parameters using the params keyword.

Change 28

There is no labeled break statement in C#. The goto is used to achieve this.

DATA TYPES


C# Data Types
Value Types
Pointers
Reference Types
Predefined Types
User-defined Types
Predefined Types
User-defined Types
Integer
Real Numbers
Booleans
Characters
Enumerations
Structures
Objects
Strings
Classes
Arrays
Delegates
Interfaces

DATA TYPES

- Value types and reference types are further classified as
  - predefined
  - User defined types
- Every variable in C# is associated with a data type.
- Data types specify the size and type of values that can be stored.
- C# is a language rich in its data types.
- The variety available allows the programmer to select the type appropriate to the needs of the application. The types in C# are primarily divided into two categories:
  - Value types
  - Reference types
- Value types and reference types differ in two characteristics:
  - Where they are stored in the memory
  - How they behave in the context of assignment statements

DATA TYPES

- Value types (which are of fixed length) are stored on the stack and when a value of a variable is assigned to another variable, the value is actually copied.
- This means that two identical copies of the value are available in memory. Reference types (which are of variable length) are stored on the heap, and when an assignment between two reference variables occurs, only the reference is copied; the actual value remains in the same memory location.
- This means that there are two references to a single value.
- A third category of types called pointers is available for use only in unsafe code.

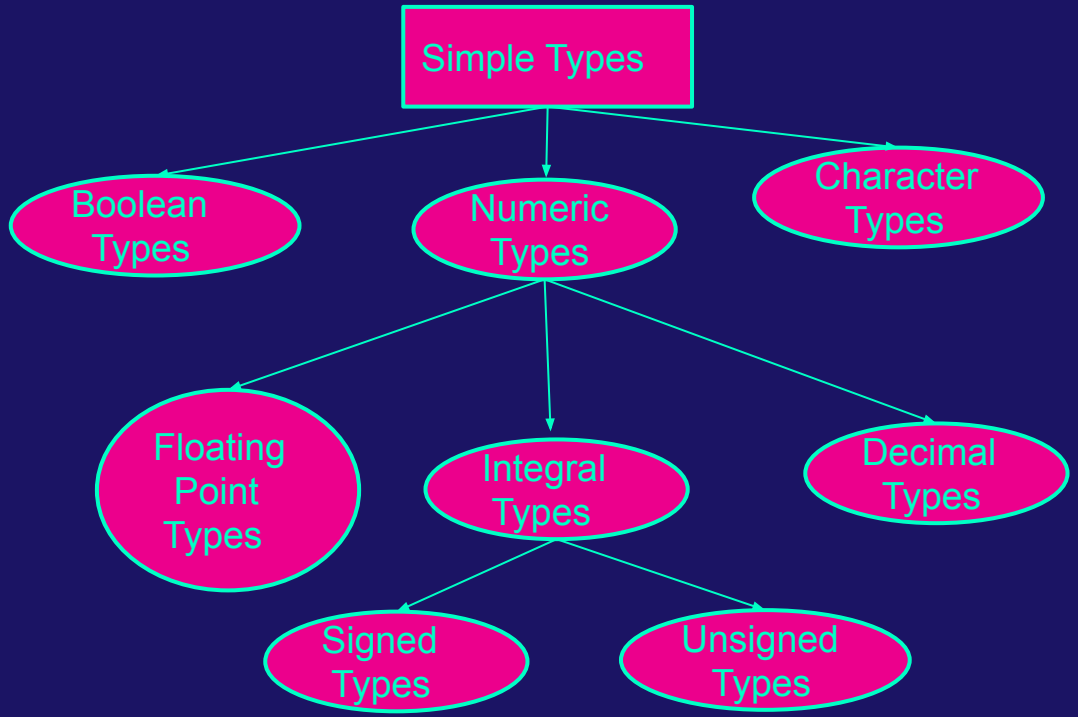
VALUE TYPES

- The value types of C# can be grouped into two categories namely,
  - 😊 User-defined types (or complex types) and
  - 😊 Predefined types (or simple types).
- Complex types known as user-defined value types can be defined which include.
  - 😊 Struct types and
  - 😊 Enumerations.
- Predefined value types which are also known as simple types (or primitive types) are further subdivided into:
  - 😊 Numeric types,
  - 😊 Boolean types, and
  - 😊 Character types.

VALUE TYPES

- Numeric types include
  - Integral types
  - Floating-point types
  - Decimal types
- C# 2.0 added a new type called nullable type.
  - This type variable can hold an undefined value.
  - Any value type variable can be defined as a nullable type.

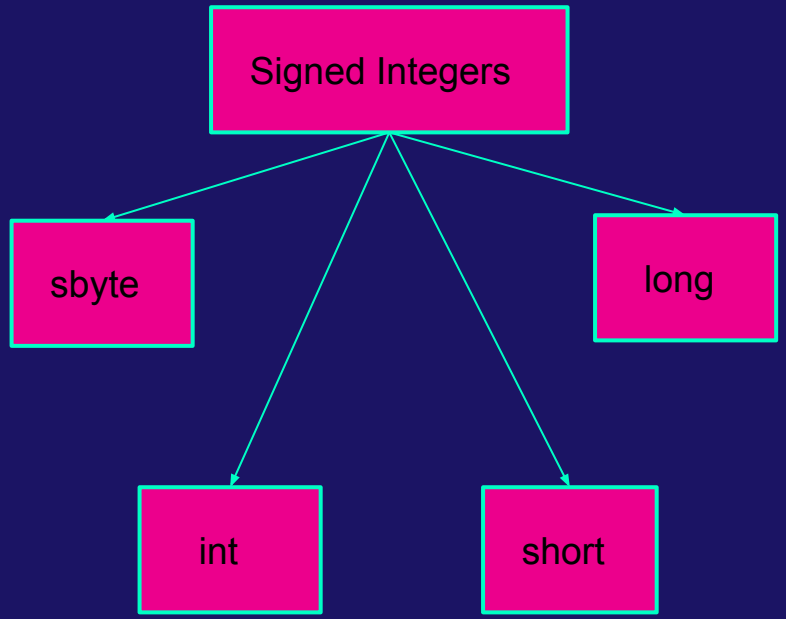
CATEGORIES OF SIMPLE TYPE DATA

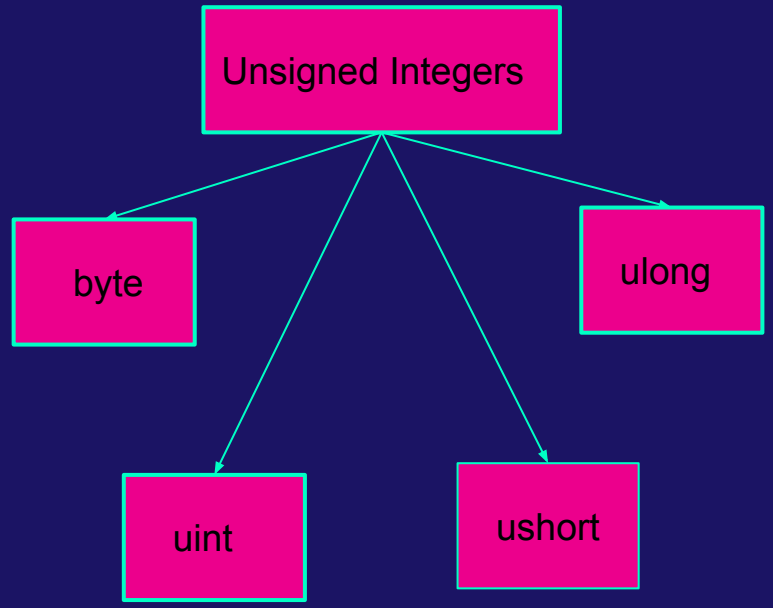


INTEGRAL TYPES

- Integral types can hold whole numbers such as 123, -96 and 5639.
- The size of the values that can be stored depends on the integral data type chosen.
- C# supports the concept of unsigned types and therefore it supports eight types of integers.

SIGNED AND UNSIGNED INTEGER TYPES





SIGNED INTEGERS

- Signed integer types can hold both positive and negative numbers.
- It should be remembered that wider data types require more time for manipulation and therefore it is advisable to use smaller data types wherever possible.
- For example, instead of storing a number like 50 in a int type variable, a sbyte variable must be used to handle this number.
- This will improve the speed of execution of the program.

SIZE AND RANGE OF SIGNED INTEGER TYPES

Type	Size	Minimum Value	Maximum Value
sbyte	One byte	-128	127
short	Two bytes	-32768	32767
int	Four bytes	-2,147,483,648	2,147,483,647
long	Eight bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

UNSIGNED INTEGERS

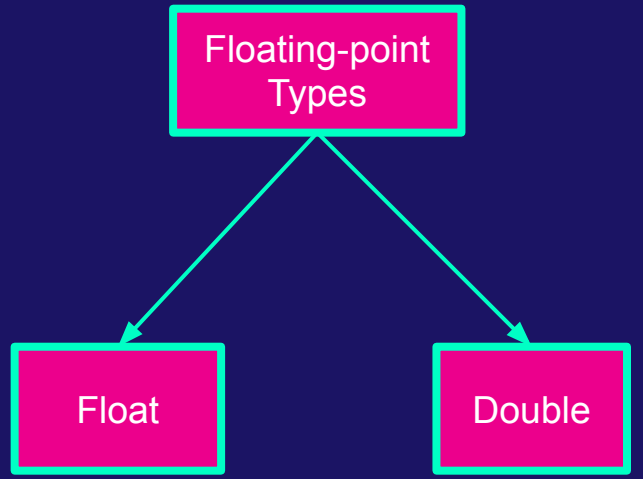
- The size of the positive value stored in an integer type can be increased by making it 'unsigned'.
- For example, a I6-bit short integer can store values in the range -32,768 to 32,767.
- However, by making it ushort, it can handle values in the range 0 to 65,535.
- All integers are by default int type.
- ln order to specify which of the other integer types the values must take, the characters U, L or UL must be appended.
- Eg. 123 U (for uint type) 123 L (for long type) 123 UL (for ulong type). Lower case u and l can also be used.

SIZE AND RANGE OF UNSIGNED INTEGER TYPES

Type	Size	Minimum Value	Maximum Value
byte	One byte	0	255
ushort	Two bytes	0	65,535
uint	Four bytes	0	4,294,967,295
ulong	Eight bytes	0	18,446,744,073,709,551,615

FLOATING-POINT TYPES

- Integer types can hold only whole numbers and therefore another type known as floating-point type is used to hold numbers containing fractional parts such as 27.59 and -1.375.
- There are two kinds of floating point storage in C#.
- The float type values are single-precision numbers with a precision of seven digits.
- The double types represent double-precision numbers with a precision of 15/16 digits.



FLOATING-POINT TYPES

- Floating-point numbers, by default are double-precision quantities.
- To force them to be in single precision mode, F must be appended to the numbers.
  - Example: l.23f 7.56923e5f
- Double-precision types are used when greater precision is needed in storage of floating-point numbers.
- Floating-point data types support a special value known as Not-a-Number (NaN).
- NaN is used to represent the result of operations such as dividing zero by zero, where an actual number is not produced.
- Most operations that have NaN as an operand will produce NaN as a result.

SIZE AND RANGE OF FLOATING POINT TYPES

Type	Size	Minimum Value	Maximum Value
float	Four bytes	1.5×10^{-45}	3.4×10^{38}
double	Eight bytes	5.0×10^{-324}	1.7×10^{308}

DECIMAL TYPE

- The decimal type is a high precision, 128-bit data type that is designed for use in financial and monetary calculations.
- It can store the values in the range 1.0×10^{-28} to 7.9×10^{28} with 28 significant digits.
- The character M or m of the value should be appended to specify the number to be decimal type.
- If M is omitted, the value will be treated as double.

CHARACTER TYPE

- C# provides a character type data called char to store single characters in memory.
- The char type assumes a size of two bytes and can hold only a single character.
- It has been designed to hold a 16-bit Unicode character, in which the 8-bit ASCII code is a subset.

BOOLEAN TYPE

- Boolean type can take two values true or false.
- Denoted by the keyword bool and uses only one bit of storage.
- No conversion between bool type and other integer types is possible.
- It is used when the user want to test particular condition during the execution of the program.

VARIABLE

- A variable is an identifier that denotes a storage location used to store a data value.
- Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program.
- Every variable has a type that determines what values can be stored in the variable. A variable name can be chosen by the programmer in a meaningful way so as to reflect what it represents in the program.
- Some examples of variable names are:
  - • average • height • total_height • classStrength

VARIABLE

Variable names may consist of alphabets, digits and the underscore (_),subject to the following conditions:

1. They must not begin with a digit.
2. Uppercase and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
3. lt should not be a keyword.
4. White space is not allowed.
5. Variable names can be of any length.

INTEGER LITERALS

- An integer literal refers to a sequence of digits.
- There are two types of integers namely,
  - Decimal integers
  - Hexadecimal integers
- Decimal integers consist of a set of digits, 0 through 9, preceded by an optional minus sign.
- Valid examples of decimal integer literals are: 123 -321 0 654321

INTEGER LITERALS

- Embedded spaces, commas and non-digit characters are not permitted between digits.
- For example, 15 750 20,000 $1000 are illegal numbers.
- A sequence of digits preceded by 0x or 0X is considered as a hexadecimal integer (hex integer).
- lt may also include alphabets A through F or 'a' through 'f'.
- A letter A through F represents the numbers 10 through 15.
- Following are the examples of valid hex integers. 0X2 0X9F 0Xbcd 0x4.

REAL LITERALS

- Integer literals are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices and so on.
- These quantities are represented by numbers containing fractional parts like 17.548 and are called real (or floating point) numbers.
- Examples of real literals are: 0.0083 -0.75 435.36
- These numbers are shown in decimal notation, having a whole number followed by a decimal point and the fractional part, which is an integer.
- It is possible that the number may not have digits before the decimal point, or digits after the decimal point.
- Examples : 215. .95 -.71 are all valid real literals.

REAL LITERALS

- A real literal may also be expressed in exponential (or scientific¡ notation. For example, the value 215.65 may be written as 2. 1565e2 in exponential notation.
- e2 means multiply by 10^2.
- The general form is: mantissa e exponent
- The mantissa is either a real number expressed in decimal notation or an integer.
- The exponent is an integer with an optional plus or minus sign.

REAL LITERALS

- The letter 'e' separating the mantissa and the exponent can be written in either lowercase or uppercase.
- Since the exponent causes the decimal point to 'float', this notation is said to represent a real number in floating-point form.
- Examples of legal floating-point literals are: 0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1
- Embedded white (blank) space is not allowed in any numeric constant.

REAL LITERALS

- Exponential notation is useful for representing numbers that are either very large or very small in magnitude.
- For example, 7500000000 may be written as 7.5E9 or 75E8.
- Similarly, -0.000000368 is equivalent to -3.68E-7.
- A floating-point literal may thus comprise four parts:
  1. Whole number
  2. Decimal point
  3. Fraction part
  4. Exponent

BOOLEAN LITERALS

- There are two Boolean literal values:
  - true
  - false
- They are used as values of relational expressions

SINGLE CHARACTER LITERALS

- A single-character literal (or simply character constant) contains a single character enclosed within a pair of single quote marks.
- Example of character in the examples above constants are: '5' 'X' ';'
- Note that the character constant '5' is not the same as the number 5.
- The last constant in the example above is a blank space.

STRING LITERALS

- A string literal is a sequence of characters enclosed between double quotes.
- The characters may be alphabets, digits, special characters and blank spaces.
- Examples are: "Hello C#" "2001" "WELL DONE" "?... !" "5+3" "X"

BACKSLASH CHARACTER LITERAL

- C# supports some special backslash character constants that are used in output methods.
- For example, the symbol '\n' stands for a new-line character.
- Each one represents one character, although they consist of two characters.
- These character combinations are known as escape sequences.

BOXING AND UNBOXING

- In object-oriented programming, methods are invoked using objects.
- Since value types such as int and long are not objects, we cannot use them to call methods.
- C# enables us to achieve this through a technique known as boxing .
- Boxing means the conversion of a value type on the stack to a object type on the heap.

BOXING AND UNBOXING

- Conversely, the conversion from an object type back to a value type is known as unboxing.
- Boxing Any type, value or reference can be assigned to an object without an explicit conversion.
- When the compiler finds a value type where it needs a reference type, it creates an object 'box' into which it places the value of the value type.

BOXING AND UNBOXING

- int m = 100;
- object om = m; // creates a box to hold m

- When executed, this code creates a temporary reference_type 'box' for the object on heap.
- A C-style cast can be used for boxing.
  - int m = 100;
  - object om= (object)m; // C-style casting
- The boxing operation creates a copy of the value of the m integer to the object om.
- Both the variables m and om exist but the value of om resides on the heap.
- This means that the values are independent of each other.

BOXING AND UNBOXING

- Consider the following code:
  - int m = 10:
  - object om = m;
  - m = 20;
  - Console.WriteLine(m); // m = 20
  - Console.Writeline(om); //om= 10
- When a code changes the value of m, the value of om is not affected.

BOXING AND UNBOXING

- Unboxing is the process of converting the object type back to the value type.
- Previously boxed variable can only be unboxed
- In contrast to boxing, unboxing is an explicit operation using C-style casting.
  - int m = 10;
  - object om = m; //box m
  - int n = (int)om; //unbox om back to an int
- When performing unboxing, C# checks that the value type requested is actually stored in the object under conversion, Only then it is unboxed

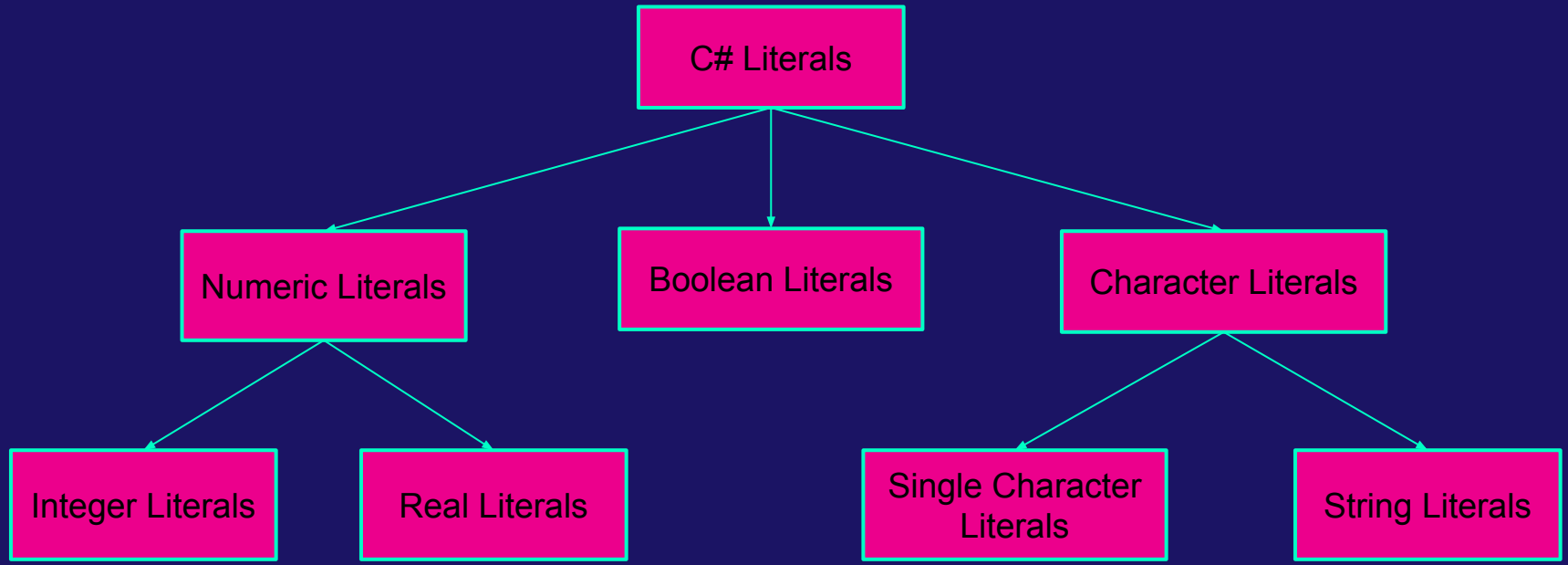
BOXING AND UNBOXING (cont.)

- When unboxing a value, ensure that the value type is large enough to hold the value of the object. Otherwise,the operation may result in a runtime error.
- For example, the following code produce a runtime error

```
int m = 500;
object om = m;
byte n = (byte)om;
```

- When unboxing, explicit casting is needed because an object could be cast to any type and the compiler verifies that it is valid as per the specified value type

C# LITERALS


C# Literals
Numeric Literals
Boolean Literals
Character Literals
Integer Literals
Real Literals
Single Character Literals
String Literals

BACKSLASH CHARACTER LITERALS

CONSTANT	MEANING
'\a'	alert
'\b'	Back space
'\f'	Form feed
'\n'	new-line
'\r'	carriage return
'\t'	Horizontal tab
'\v'	Vertical tab
'\''	Single quote
'\"'	Double quote
'\\'	backslash
'\o'	null

ARITHMETIC OPERATORS

- C# provides all the basic arithmetic operators.
- The operators can operate on any built-in numeric data type.
- Cannot use operators on Boolean type.

OPERATORS	MEANING
+	Addition or Unary Plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo Division

TYPES OF ARITHMETIC OPERATORS

Integer Arithmetic

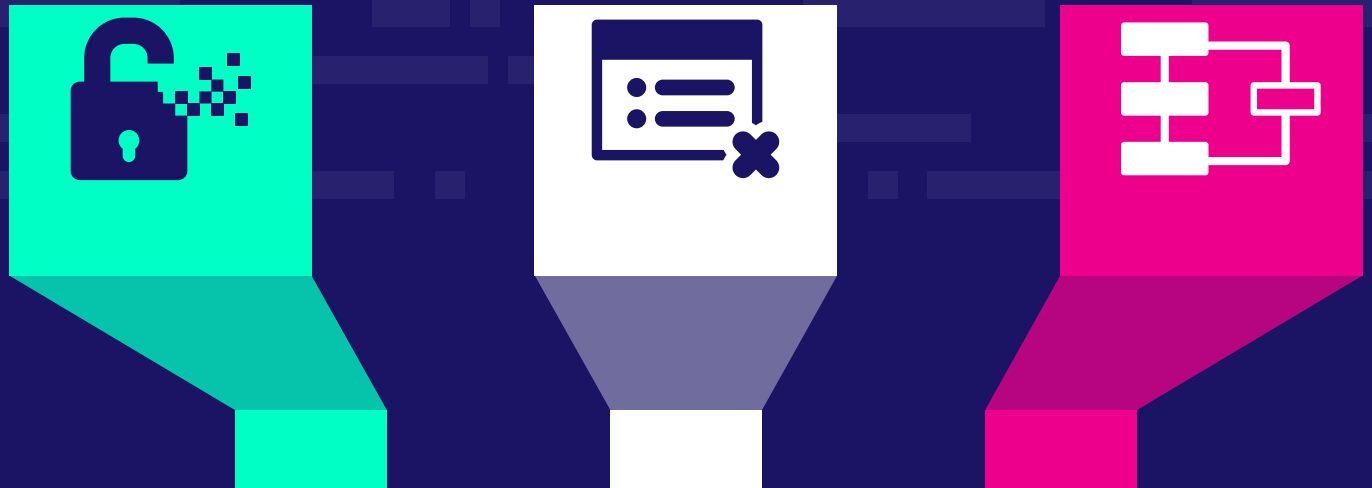
When both the operands in a single arithmetic expression are integers it is called arithmetic expression and the operation integer arithmetic.

Real Arithmetic

Involving only real operators. A real operand assume values either in decimal or exponential notation

Mixed-Mode Arithmetic

When one of the operand is real and the other is integer it is called mixed-mode arithmetic expression.



LOGICAL OPERATORS

- An expression which combines two or more relational expressions is termed as logical expression or a compound relational expression.

OPERATORS	MEANING
&&	Logical AND
||	Logical OR
!	Logical NOT
&	Bitwise logical AND
|	Bitwise Logical OR
^	Bitwise Logical Exclusive OR

TRUTH TABLE

op-1	op-2	Value of the Expression	
		Op-1 && op-2	Op-1 || op-2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

ASSIGNMENT OPERATORS

- Assigns the value of an expression to a variable.
- The assignment operator is =.
- The general form of shorthand assignment operator is
  - v op = exp is equivalent to
  - v= v op(exp).

STATEMENT WITH SIMPLE ASSIGNMENT OPERATORS	STATEMENT WITH SHORTHAND OPERATOR
a=a+1	a+=1
a=a-1	a-=1
a=a*(n+1)	a*=n+1
a=a/(n+1)	a/=n+1
a=a%b	a%=b

ASSIGNMENT OPERATORS

- v is a variable
- exp is an expression.
- op is a C# binary operator.
- The operator op= is called the shorthand assignment operator.

INCREMENT AND DECREMENT OPERATORS

INCREMENT OPERATOR

The increment operator ++
adds 1 to the operand

DECREMENT OPERATOR

The Decrement operator --
subtracts 1 from the operand

CONDITIONAL OPERATOR

?:

The ternary operator ?: is used to construct conditional expressions of the form

exp1? exp2 : exp3

Where exp1,exp2,exp3 are expressions.

eg:

```
a=10;
b=15;
x=(a>b)?a:b;
```

CONDITIONAL OPERATOR

The ternary operator ?: is used to construct conditional expressions of the form

exp1? exp2 : exp3

Where exp1,exp2,exp3 are expressions.

eg:

```
a=10;
b=15;
x=(a>b)?a:b;
```

BITWISE OPERATORS

- Operators used for manipulation of data at bit level.
- Used for testing the bits or shifting them right or left,
- Cannot be applied to floating point data.

OPERATORS	MEANING
&	Bitwise logical AND
|	Bitwise logical OR
^	Bitwise logical XOR
~	One's complement
<<	Shift left
>>	Shift right

SPECIAL OPERATORS

Operator	Meaning
is	Relational operator
as	Relational operator
typeof	type operator
sizeof	size operator
new	Object creator
.(dot)	Member-access operator
checked	Overflow checking
unchecked	Prevention of overflow checking

ARITHMETIC EXPRESSIONS

- A combination of variables, constants and operators arranged as per the syntax of the language.

ALGEBRAIC EXPRESSION	C# EXPRESSION
a×b-c××	a*b-c
(m+n)(x+y)	(m+n)*(x+y)
ab÷c	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1
x÷y+c	x/y+c

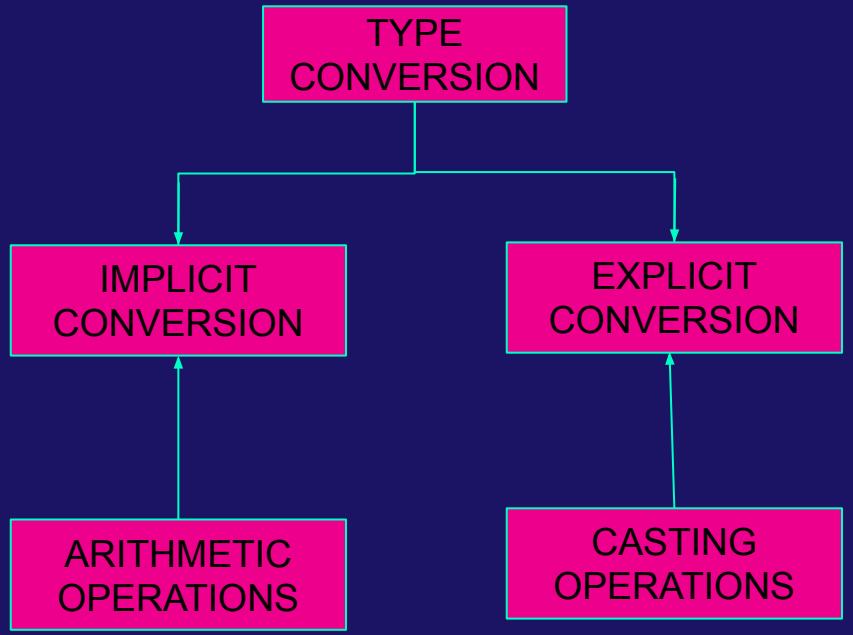
Evaluation of Expressions

- Expressions are evaluated using an assignment statement of the form
- variable=expression
- Variable is any valid C# variable names.
- When the statement is encountered the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.
- All variables used in the expression must be assigned values before evaluation is attempted.

TYPE CONVERSIONS

In C# type conversions take place in two ways.

- Implicit conversions
- Explicit conversions



Implicit Conversions

Conversion that can be performed without any loss of data. Also known as automatic type conversion

Explicit Conversions

Explicitly carry out conversions using cast operator. The process is known as casting.

type variable1= (type) variable2;

MATHEMATICAL FUNCTIONS

METHOD	DESCRIPTION
Sin()	Sine of an angle in radians
Cos()	Cosine of an angle in radians
Tan()	Tangent of an angle in radians
Asin()	Inverse of sine
Acos()	Inverse of cosine
Atan()	Inverse tangent with x and y coordinates specified
Atan2()	Hyperbolic sine
Sinh()	Hyperbolic cosine
Cosh()	Hyperbolic tangent

MATHEMATICAL FUNCTIONS

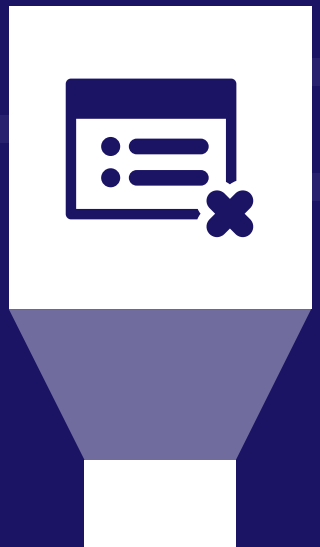
METHOD	DESCRIPTION
Tanh()	Hyperbolic tangent
Sqrt()	Square root
Pow()	Number raised to a given power
Exp()	Exponential
Log()	Natural logarithm (base e)
Log10()	Base 10 logarithm
Abs()	Absolute value of a number
Min()	Lower of two numbers
Max()	Higher of two numbers
Sign()	Sign of a number

Decision Making and Branching

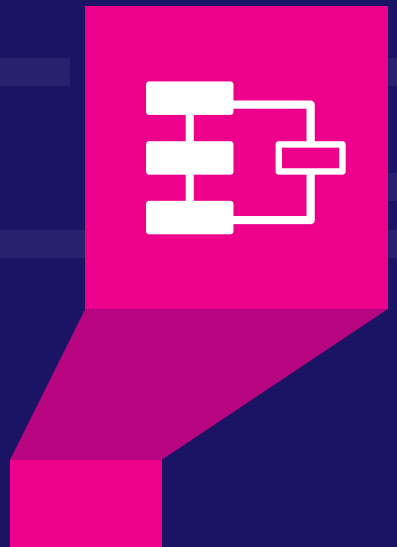
CONTROL OR DECISION MAKING STATEMENTS

IF Statement

Switch Statement



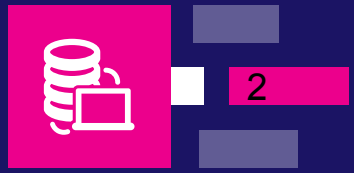
Conditional Operator Statement

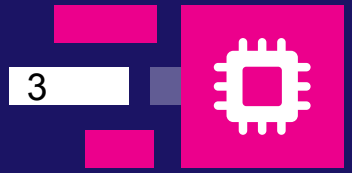


DIFFERENT FORMS OF IF STATEMENTS

Simple if

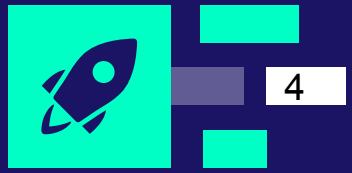
if..else





Nested
if..else

Else if
ladder



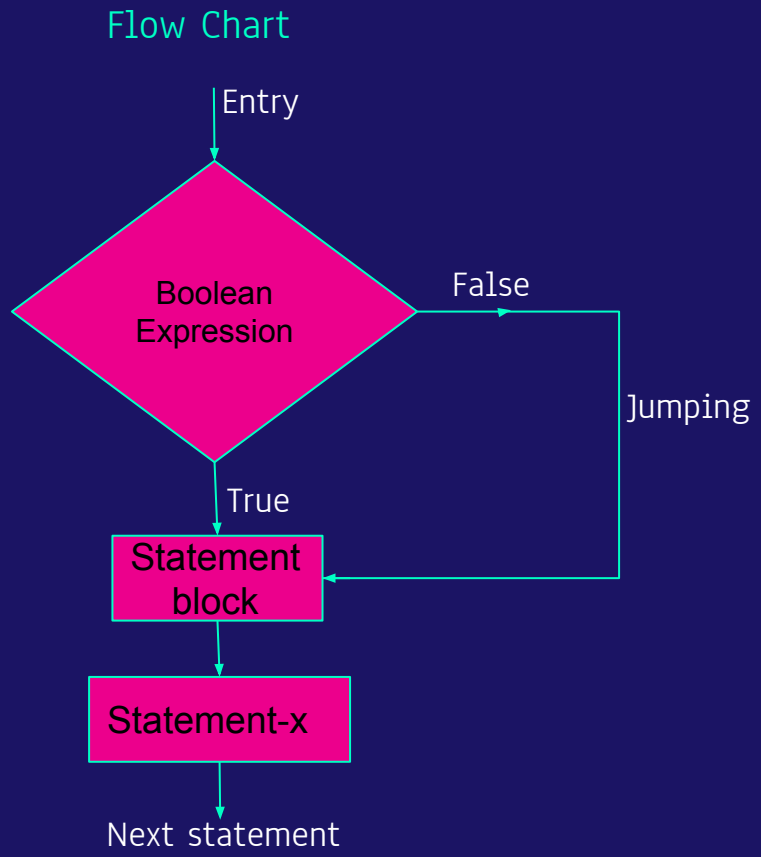
SIMPLE IF STATEMENT

The general form of a simple if statement is

```
if(boolean
expression)
 {
   Statement-block;
 }
 Statement-x;
```

SIMPLE IF STATEMENT

- The 'statement-block' may be a single statement or a group of statements.
- If the boolean-expression is hue, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x.



IF...ELSE STATEMENT

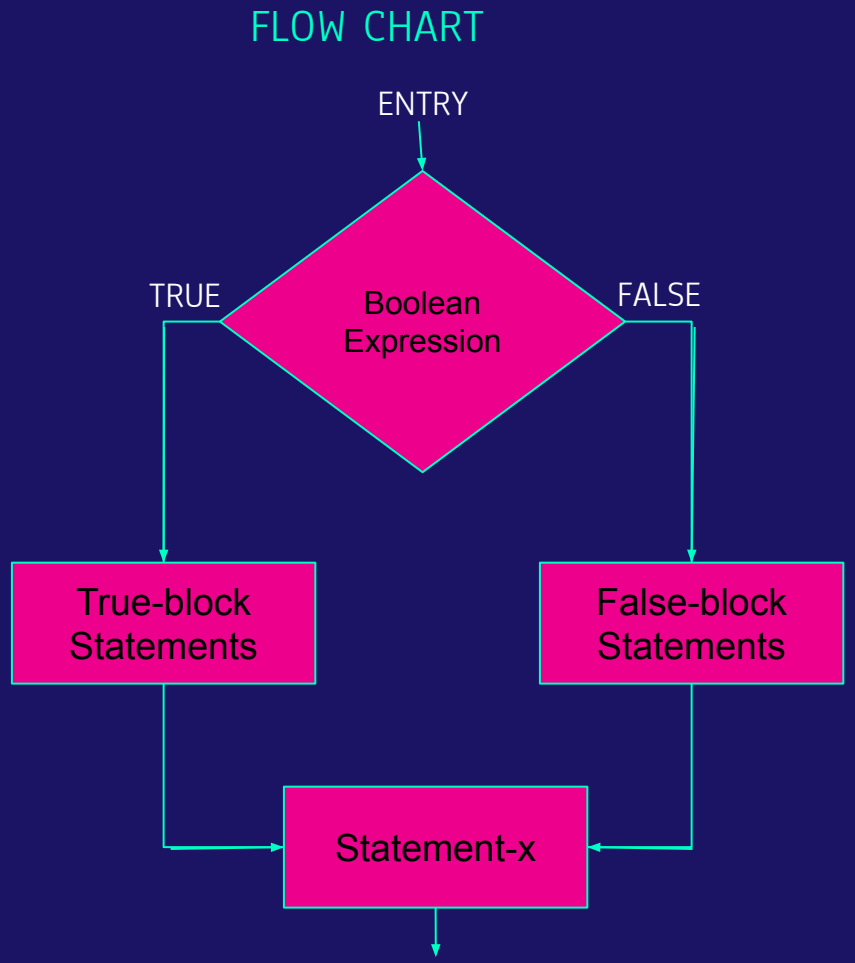
The if.,..else statement is an extension of the simple if statement.

The general form is

```
if(boolean_expression)
  {
   True-block statement(s)
  }
else
  {
   False-block statement(s)
  }
statement-x
```

IF...ELSE STATEMENT

- If the boolean expression is true, then the true-block statements, immediately following the if statement, are executed; otherwise, the false-block statements are executed.
- In either case, either true-block or false block will be executed, not both.



NESTING OF IF...ELSE STATEMENTS

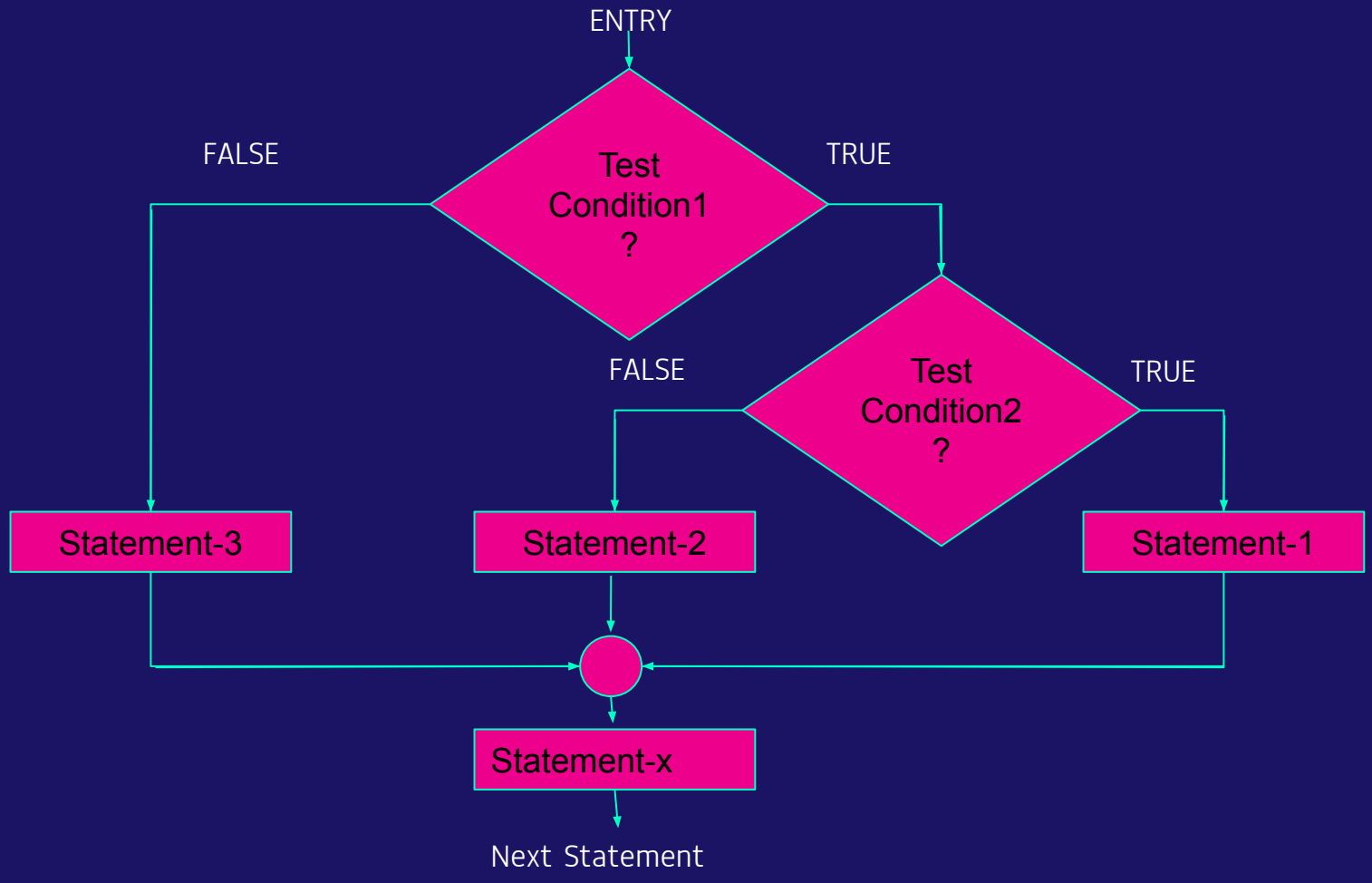
```
if (test condition1)
 {
    if {test condition2)
    {
        Statement-1;
    }
    else
    {
        statement-2;
    }
 }
else
 {
    statement-3;
 }
statement-x;
```

NESTING OF IF...ELSE STATEMENTS

The logic of execution

- If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second test.
- If the condiüon-2 is true, then statement-1 will be evaluated; otherwise statement-2 will be evaluated and the control transferred to statement-X.

NESTING OF IF...ELSE STATEMENTS



THE ELSE IF LADDER

```
if (condition 1)
    statement-1;
else if (condition 2)
    statement-2;
    else if (condition 3)
        statement-3;
    else if (condition n)
        statement-n;
    else
        default-statement;
 statement-x;
```

THE ELSE IF LADDER

- A multipath decision is a chain of ifs in which the statement associated with each else is an if is known as the else if ladder.
- The conditions are evaluated from the top (of the ladder), downwards.
- As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder).
- When all the n conditions become false, then the final case containing the default-statement will be executed.

THE SWITCH STATEMENT

The general form of switch statement

```
switch(expression)
{
     case value-1:
          block-1
          Break;
     case value-2:
          block-2
          break;
     ……
     ……
     default:
          default-block
          break;
}
 statement-x;
```

THE SWITCH STATEMENT

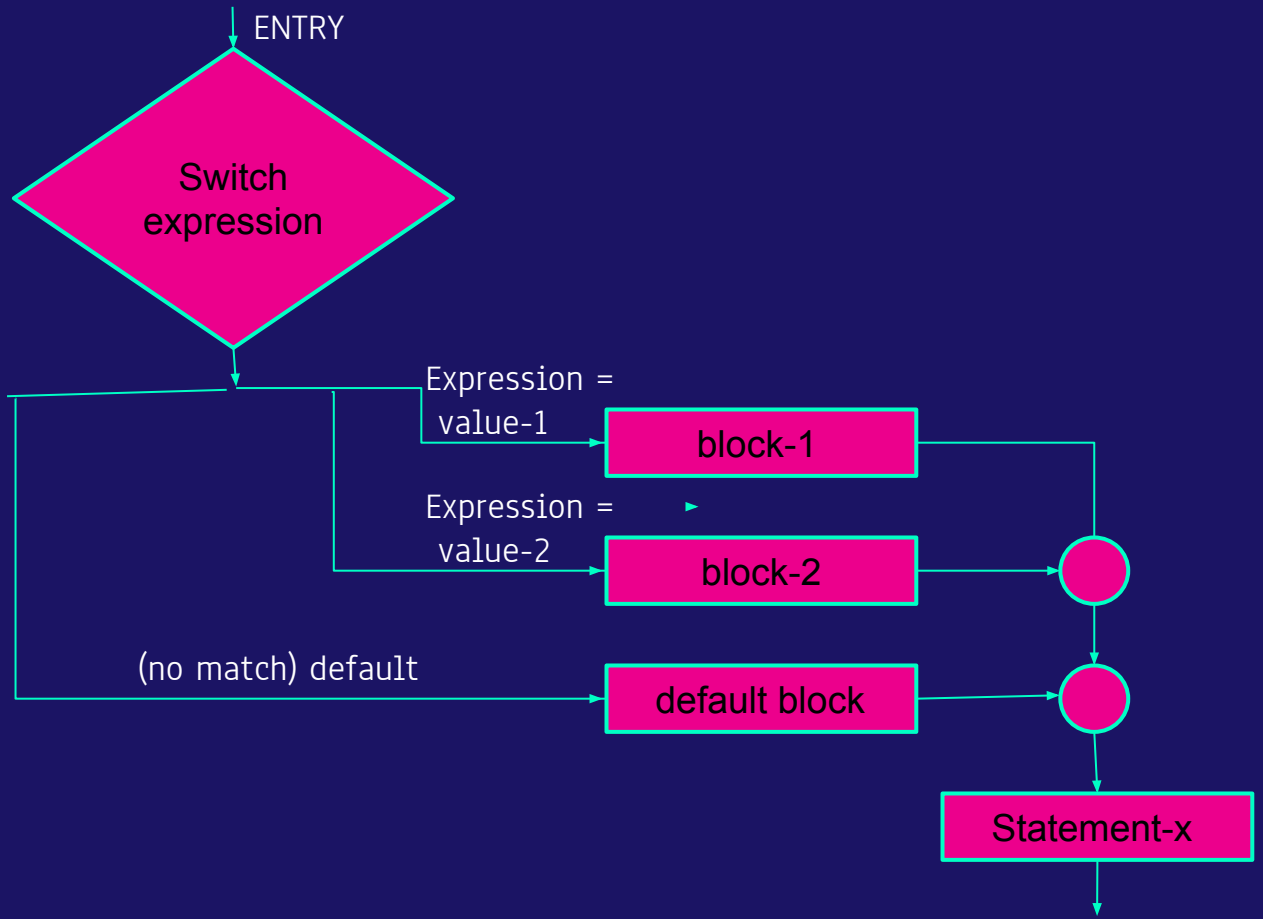
- The expression must be an integer type or char or string type.
- value-1, value-2 .... are constants or constant expressions and are known as case labels.
- Each of these values should be unique within a switch statement.
- block-1, block-2 .... are statement lists and may contain zero or more statements.
- There is no need to put braces around these blocks but it is important to note that case labels end with a colon(:).

THE SWITCH STATEMENT(cont.)

The switch statement is executed in the following order:

1. The expression is evaluated first.
2. The value of the expression is successively compared against the values, value-1, value-2, ... If a case is found whose value matches the value of the expression, then the block of statements that follows the case are executed.
3. The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statement-x following the switch.
4. The default is an optional case. When present, it will be executed if the value of the expression does not match any of the case values. If not present, no action takes place when all matches fail and the control goes to the statement-x.

SWITCH STATEMENT FLOWCHART



Decision Making and Looping

LOOPING

- The process of repeatedly executing a block of statements is known as looping.
- The statements in the block may be executed any number of times, from zero to an infinite number.
- If a loop continues forever, it is called an infinite loop.
- C# supports such looping features that enable us to develop concise programs containing repetitive processes without using unconditional branching statements like the goto statement.
- ln looping, a sequence of statements are executed until some conditions for the termination of the loop are satisfied.
- A program loop therefore consists of two segments, one known as the body of the loop and the other known as the control statement.
- The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

LOOPING

- Depending on the position of the control statement in the loop, a control structure may be classified as
  - the entry-controlled loop
  - the exit-controlled loop.
- ln the entry-controlled loop, the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed.
- ln the case of an exit-controlled loop, the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time.

LOOPING

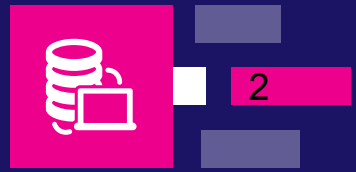
- The test conditions should be carefully stated in order to perform the desired number of loop executions.
- lt is assumed that the test condition will eventually transfer the control out of the loop.
- If for some reason it does not do so, the control sets up an infinite loop and the body is executed over and over again.
- A looping process, in general, would include the following four steps:

1. Setting and initialization of a counter.
2. Execution of the statements in the loop.
3. Testfor a specified condition for execution of the loop.
4. Incrementing the counter.

CONSTRUCTS FOR PERFORMING LOOP OPERATIONS

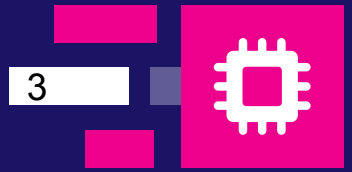
The While Statement

Entry-controlled loop statement



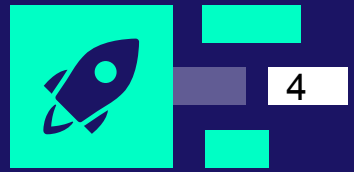
The do statement

Exit-controlled loop statement



The for statement

Entry-controlled loop statement



The foreach statement

Entry-controlled loop statement

THE WHILE STATEMENT

The basic format of the while statement

```
initialization;
while(test condition)
   {
    Body of the loop
   }
```

THE WHILE STATEMENT

- while is an entry-controlled loop statement.
- The test condition is evaluated and if the condition is true, then the body of the loop is executed.
- After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop.
- On exit, the program continues with the statement immediately after the body of the loop.
- The body of the loop may have one or more statements.
- The braces are needed only if the body contains two or more statements and it is a good practice to use braces even if the body has only one statement.

THE DO STATEMENT

The basic format of the do statement

```
initialization;
do
    {
     Body of the loop
    }
while(test condition);
```

THE DO STATEMENT

- On reaching the do statement, the program proceeds to evaluate the body of the loop first.
- At the end of the loop, the test condition in the while statement is evaluated.
- If the condition is true, tbe program continues to evaluate the body of the loop once again.
- This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.
- Since the test condition is evaluated at the bottom of the loop, the do....while construct provides an exit-controlled loop and therefore the body of the loop is always executed at least once.

THE FOR STATEMENT

The general form of the for loop is

```
for(initialization;test
    condition;increment)
{
   Body of the loop
}
```

THE FOR STATEMENT

The execution of the for statement is as follows:

1. Initialization of the control variables is done first, using assignment statements such as i= I and count= O. The variables i and count are known as loop-control variables.
2. The value of the control variable is tested using the test condition.
    - The test condition is a relational expression, such as i< IO, which determines when the loop will exit.
    - If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.

THE FOR STATEMENT(cont.)

3. When the body of the loop is executed, the control is transferred back to the for statement after evaluating the last statement in the loop.
    - Now, the control variable is incremented using an assignment statement such as i = i+I and the new value of the control variable is again tested to sec whether it satisfies the loop condition.
    - If the condition is satisfied, the body of the loop is again executed. This process continues till the value of the control variable fails to satisfy the test condition.

THE FOREACH STATEMENT

The general form of foreach statement

```
foreach(type variable
        in expression)
{
Body of the loop
}
```

THE FOREACH STATEMENT

- The type and variable declare the iteration variable.
- During execution, the iteration variable represents the array element (or collection element in case of collections) for which an iteration is currently being performed.
- in is a keyword.
- The expression must be an array or collection type and an explicit conversion must exist from the element type of the collection to the type of the iteration variable.

FOREACH STATEMENT

```
Using System;
Class ForeachTest
{
     public static void Main()
     {
          int[] arrayInt={11,22,33,44};
          foreach (int m in arrayInt)
          {
               Console.Write(“”+m);
          }
               Console.WriteLine();
     }
}
 Output
11 22 33 44
```

THANKS !

Do you have any questions?
buvann@gmal.com

Credits

- E.Balagurusamy , "Programming in C#" , Third Edition, Tata McGraw Hill Education Private Limited, New Delhi.
- www.slidesgo.com
- https://www.brainbell.com/tutors/C_Sharp/Microsoft_Intermediate_Language_and_the_JITters.htm