About C#

- Two languages that are central to .NET framework are C# and Visual Basic.
- It is fundamentally a simple and object oriented language.
- It was developed by Microsoft development team.
- Lot of syntax for C# is obtained C language. C# also shares many similarity with popular Java language, thereby making learning of C# even more simpler.
- Several version of C# have come up starting from 2002. The latest version is C# 10.0. Visual Studio 2022 supports this.

Comments

Single line comment

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

Multiline comments

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

Variables

- Variables are storage locations that are associated with a value.
- The value that a variable stores are of certain data type.
- **Syntax: Datatype var_name;**
- There are different types of variables that can be defined in C# depending on the scope : static variables, instance variables, array elements, and local variables.
- The variables created inside Method are local. They are available only inside the method where they are declared.
- The local variable must always be initialized before using.

Keywords

abstract	event	new	struct	decimal
as	explicit	null	switch	default
base	extern	object	this	int
bool	false	operator	throw	double
break	finally	out	true	else
byte	fixed	override	try	enum
case	float	params	typeof	lock
catch	for	private	uint	long
char	foreach	protected	ulong	namespace
checked	goto	public	unchecked	stackalloc
class	if	readonly	unsafe	static
const	implicit	ref	ushort	string
continue	in	return	using	internal
delegate	interface	sbyte	virtual	is
do	short	sealed	volatile	void
sizeof	while			

C# Data types

- C# like C and C++ is a strongly typed language and so every variable must have a data type.
- Data types in C# are of 2 types
  - Value types
  - Reference types.

C# Data types

- Value types
  - Built in Data types like int, char, float ,double and user defined types like struct or enum
  - For every value type there is a type(Class or Struct) in BCL
  - They are allocated in stack
  - Value types can not contain the value null.

C# Data types

- Reference
  - User defined type created using class or interfaces
  - Reference types are allocated in heap at runtime.
  - Reference types can contain the value null

C# Value Types

Built in Types

- Boolean type
  - `bool`
- Integer types
  - `byte int long sbyte short uint ulong ushort`
- Character type
  - `char`
- Floating point types
  - `decimal double float`

User Defined Types

- `enum`
- `struct`

Integer types

C# type	CLS compliant	System Type	Range
`sbyte`	No	`System.Sbyte`	-128 to 127 (signed 8-bit)
`byte`	Yes	`System.Byte`	0 to 255 (unsigned 8 bit)
`short`	Yes	`System.Int16`	-32768 to 32767 (signed 16 bit)
`ushort`	No	`System.UInt16`	0 to 65535 (unsigned 16 bit)
`int`	Yes	`System.Int32`	-2147783648(-2^{31}) to 2147483647(2^{31}-1) (signed 32 bit)
`uint`	No	`System.UInt32`	0 to 2^{32}-1 (unsigned 32 bits)
`long`	Yes	`System.Int64`	-2^{63} to 2^{63}-1 (signed 64 bit number)
`ulong`	No	`System.UInt64`	0 to 2^{64}-1 (unsigned 64 bit number)

Floating point types

C# type	CLS compliant	System Type	Range
`float`	Yes	`System.Single`	$1.5*10^{-45}$ to $3.4*10^{38}$ (32 bit floating point number)
`double`	Yes	`System.Double`	$5.0*10^{-324}$ to $1.7*10^{308}$ (64 bit floating point number)
`decimal`	Yes	`System.Decimal`	±1.0 x 10−28 to ±7.9 x 1028 (28-29 significant digits)

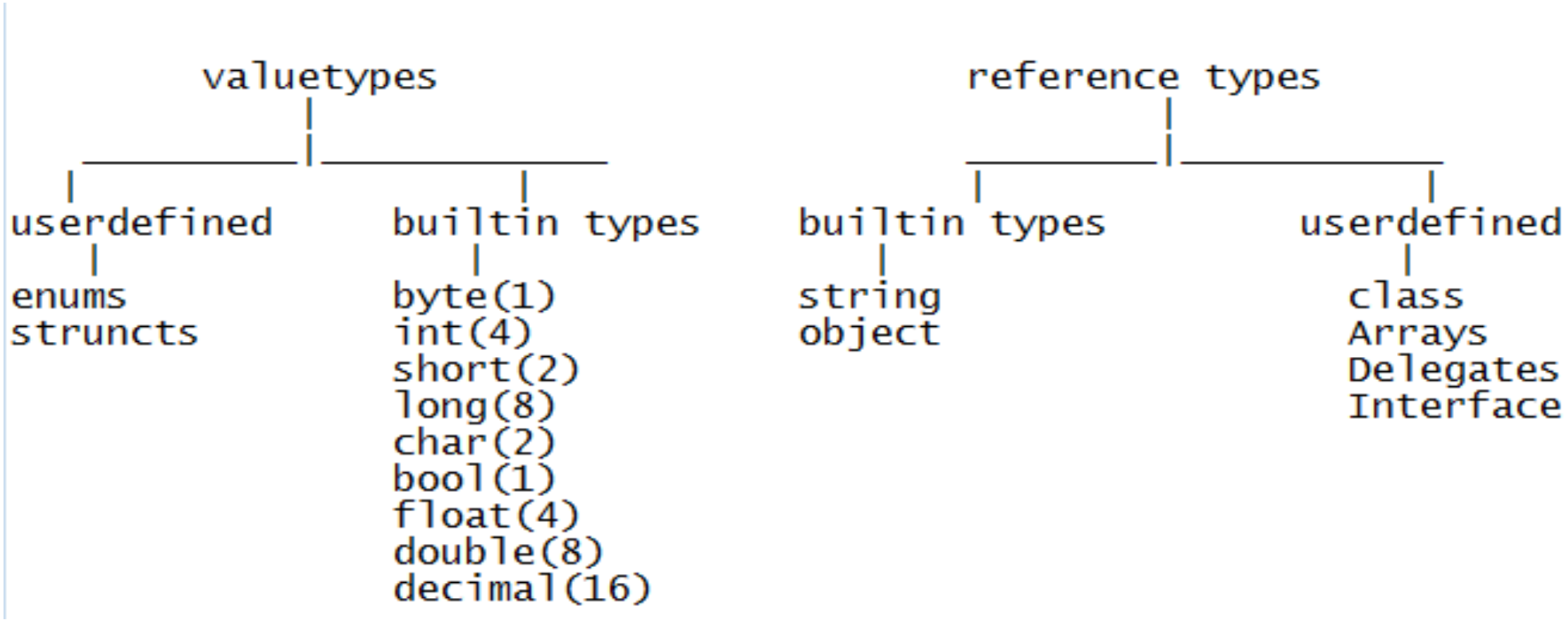
Floating-point literals

- Floating-point literals can be written in two ways
  - Fixed notation: 3.14
  - Scientific notation: 0.314E1, 314e-2
- Floating-point literal types
  - Literal with no suffix→ is double
  - Literal with F or f as suffix (Example 3.14f, 3.14F)→ is **`float`**
  - Literal with D or d as suffix (Example 3.14D, 3.14D) → is **`double`**
  - Literal with M or m as suffix (Example 3.14m, 3.14M) → is **`decimal`**

```
decimal d = 3.45; //error
decimal d = 3.45m; //ok
float f = 3.4;// error
float f = 3.4f;// ok
```

Character type and Character Literal

- **`char`** keyword is alias for **`System.Char`**
- It is CLS compliant.
- **`char c='a'; //ok`**
- **`char c=1; //error`**


valuetypes
userdefined
enums
structs
builtin types
byte(1)
int(4)
short(2)
long(8)
char(2)
bool(1)
float(4)
double(8)
decimal(16)
reference types
builtin types
string
object
userdefined
class
Arrays
Delegates
Interface

Identifiers Naming Rules

- Identifiers are variables, methods , class or any other named constructs in C#
- Rules
  - Should not be a keyword
  - Must start with a letter or underscore or @
  - Subsequent characters can be a letter, number or underscore

Operators

Operators-Arithmetic

+	-	/	*	++	--	%

- Most of the operators work the same way as it does in C.
- Applicable to integer and floating point types
- ```
  int i = 10;
  int j = i+ 255;
  byte k = i + 255; //error
  ```
- ```
  const double  PI = 3.14;
   double f = PI + 12;
  ```
- When 2 of any integral types (other than long) are involved in arithmetic operation, the result is an int.

```
byte i = 10;
byte j = 20;
j=j+i; //error
```

Operators-Relational

==	!=	<	>	>=	<=

- Relational operator return bool value – true or false.

```
int i=10;
 int j=20;
 Console.WriteLine( i>j ); // output is False
 Console.WriteLine(i == 10); // output is True
```

Operators-Logical

&&	||	!

- These are both boolean operators. These work only on bool values.
- **&&** checks if the first condition is false. If it is so then it doesn't evaluate the second condition.
- **||** checks if the first condition is true. If it is so then it doesn't evaluate the second condition.

```
int i=0;
 int j=2;
 bool b= (i>j) && (j++>i);
Console.WriteLine(j);
```

What does the code print?

Assignment Operators

=	+=	-=	*=	/=	%=

```
int a = 10;
 int b=2;
 a+=b; //  12 same as a=a+b;
 Console.WriteLine(a);

 double d=45.3;
 d/=1.2;
 Console.WriteLine(d); // 37.75 same as d=d/1.2

 float d1=2.4f;
 d1 %= 1.2f;
 Console.WriteLine(d1); // 0
```

Other Operators

Operator	Name	Example
`?:`	Ternary operator	10 % 2 > 0 ? 1 : 10
`[]`	Array subscript	a[1]
`()`	Cast	
`.`	dot operator	n.Length
`is`	Type Conversion	
`as`		
`typeof`	In refection	
`new`		
`checked`		
`unchecked`		
`=>`	Lambda	

Conversion

Conversions

- Type conversion is a process of converting values from one data type to another data types
- Conversion types:
  - Implicit Conversion
  - Casting
  - Boxing
  - Un Boxing

Implicit Conversions

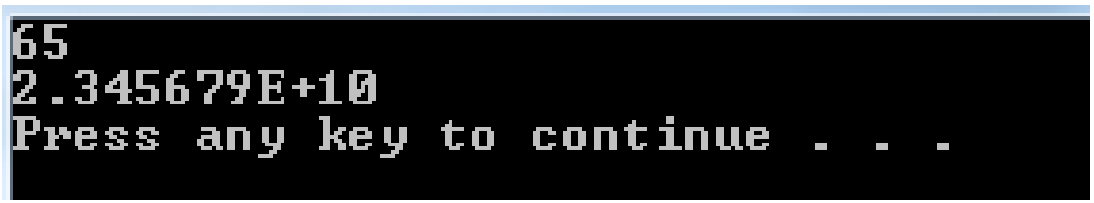
Implicit numerical conversions

- **byte → short, int, long, float, double, decimal**
- **char →int, long, float, double, decimal**
- **short →int, long, float, double, decimal**
- **int →long, float, double, decimal**
- **long →float, double, decimal**
- **float → double**

Example

```
using System;
class Convert{
public static void Main(){
char c='A';
int i=c;
Console.WriteLine(i);
long l=23456789100;
float f=l;
Console.WriteLine(f);
}}
```

vice versa is not allowed implicitly



Casting

- Any conversion that happens in the opposite direction of implicit numerical conversions requires explicit request from the compiler through casting.
- Casting
  - Number casting example

```
long l2 = 12500;

int i2 = (int)l2;
```

  -
```
int i = 256;

byte b1 = (byte)i;
```

Casting

- floating point cast to int example

```
float f=12.f;

int i2 = (int)f;
```

```
Example 1:
      float b1 = 1f;       int b2 = 2;
      float b3 = b1 + b2;
Example 2:
      long b1 = 1;              double b2 = 2;
      double b3 = b1 + b2;
Example 3:
      double b2 = 2;
      int a3 = (int)b2;
```

Convert

- Note that conversions of numeric value to string and vice versa, bool to string and vice versa etc. is not possible either implicitly or explicitly.
- This can be done through the methods below:

Example:

```
 /* Boolean values */
ii= Convert.ToBoolean("true");
      Console.WriteLine(ii); //True
      ii = Convert.ToBoolean("gg");
         Console.WriteLine(ii); // Runtime error call
                                //FormatException
```

Convert

```
/* String values */
     string s= Convert.ToString(true);
     Console.WriteLine(s); //True
      s = Convert.ToString(1.23);
      Console.WriteLine(s); //1.23
      s = Convert.ToString('A');
      Console.WriteLine(s); //A
      Console.WriteLine(Convert.ToString(null));
     // prints nothing
   /* Numeric Values */
      int i = Convert.ToInt32("1009");
      short j = Convert.ToInt16("100");
      long k = Convert.ToInt64("9292929");
      byte b=Convert.ToByte("20");
       No ToFloat method!
      double d=Convert.ToDouble("sss"); // runtime
error
```

Parse() Method

- Another way to convert strings to basic type is by using System Type
- Every System Type has Parse() method that can be used to convert a string into the respective System Type.
- <Type>. Parse(string s)

Where <Type> could be any System Type/ C# type that we looked at earlier.

```
using System;
class ParseTest{
  static void Main(string[] args)  {
      sbyte i1= sbyte.Parse("123");
      float f1=  float.Parse("123.3");
      decimal d1= decimal.Parse("123.45");
      bool b= bool.Parse("true");
Console.WriteLine("{0}, {1}, {2}, {3}, {4}, {5}, {6}", i1, f1, d1, b);
}}
```

Boxing And Unboxing

- Boxing and unboxing allows a value-type to be converted to and from type object.
- Converting a value type to reference type is called Boxing. Boxing is implicit conversion.

```
int i = 1;
object o = (object)i; // boxing
```

Or simply `object o=i;`

- And reverse is unboxing

```
o = 1;
i = (int)o; // unboxing
```

- Unboxing is explicit conversion.
- When value-type are boxed they are stored in heap.
- Boxing and unboxing are computationally expensive processes.

Conditional Statement & Loops

Decision constructs: if statement

- `if` statement

```
if(Condition)
{
statements;
}
```

Condition must evaluate to bool value

Decision constructs: if-else statement

- **if/else** statement

  if(*Condition*) ← Condition must evaluate to bool value

  { *statements;* }

  [

  else{

  statements;

  }

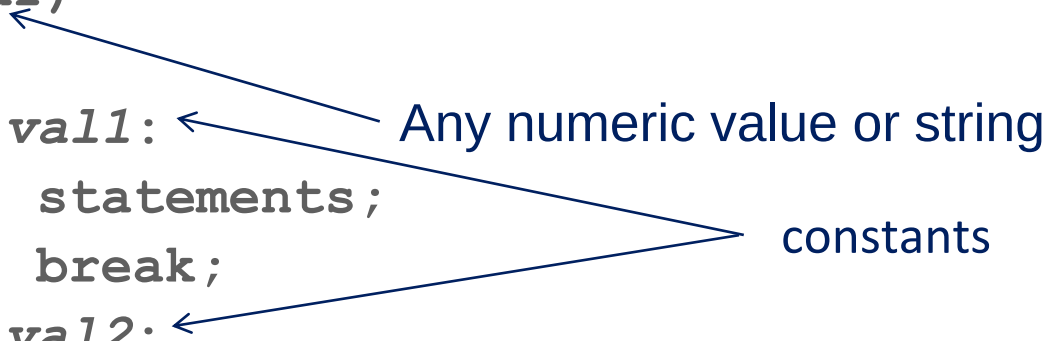
  ]

- Example: **if (c > 0) c++; else c--;**

Decision constructs: switch statement

- **switch** statement

```
switch(var)
{
    case val1:
          statements;
          break;
    case val2:
          statements;
          break;
    ……
    default: statements;
    break;       }
```



Example

```
char c;
 c = char.Parse(Console.ReadLine());
  switch(c){
      case 'r': Console.WriteLine("RED");
                  break;
      case 'g': Console.WriteLine("GREEN");
                break;
      case 'b': Console.WriteLine("BLUE");
                          break;}
    }
```

- C# compiler enforces that every case statement must have a `break` statement. In other words control cannot fall through from one case label to another.

Multiple Cases

```
char c;
        c = char.Parse(Console.ReadLine());
  switch(c){
      case 'r':
      case 'R':Console.WriteLine("RED");
                          break;
      case 'g':
      case 'G': Console.WriteLine("GREEN");
                  break;
      case 'b':
      case 'B': Console.WriteLine("BLUE");
                          break;}
    }
```

Looping Statements- for statement

There are two forms of **`for`** statement

1. for loop

```
for(intialization; condition;increment/decrement){
        //statements
  }
```

Ex: `for(int i=0;i<10;i++)`

2. foreach loop:

```
foreach(object variable in itemlist){
        //statements
}
```

Ex:
```
int[] arr={1,2,3,4,5};
 foreach(int s in arr)
        Console.WriteLine(s);
```

while and do while

```
// while loop
   while(Condition) {
       statements
   }
Ex: int i=0;
   while(i<10) {
       Console.WriteLine(i);
   i++;       }
```

Condition must evaluate to bool value

```
do-while loop
   do{
       statements
   }while(Condition);
Ex: int i=0;
   do  {
              Console.WriteLine(i);
              i++;
       } while(i<10);
```

`break` and `continue`

- Used with loop statements
- `break` is used to break out of the loop. Is used with switch statement also.
- `continue` is used to exit out of current iteration and continue with the next iteration.

```
int j = 100;
      while (true)
      {
          if (j % 13 == 0) break;
          else
          j++;

      }
      Console.WriteLine(j); //104
```