C# .net Notes

1. Assembly: C# source code is compiled into an intermediate language. These code and resources are stored in an executable file called an assembly (with an extension of .exe or .dll). An assembly contains a manifest that provides information about the assembly’s types, version, culture, and security requirements. When the C# program is executed, the assembly is loaded into common language runtime.
2. Visual studio, comment code: ctrl K+ ctrl c, uncomment code: ctrl k + ctrl u
3. byte – short – int – long(8-16-32-64 bit integer)
4. float – double -decimal (7 digits fraction-15-28), we use decimal for counted values and use float/double for measured values (https://stackoverflow.com/questions/618535/difference-between-decimal-float-and-double-in-net)
5. enum Day{Sat, Sun, Mon, Tue, Wed, Thu, Fri}
6. define a default constructor for a struct is an error, struct can be instantiated without using new operator. You must initialize all members when write a constructor with parameters. It cannot be inherited.
7. Unmanaged type
8. Abstract class cannot be instantiated; sealed class prevented it from being inherited. Non-abstract class derived from an abstract class must implemented all abstract members. Abstract and interface
9. An abstract method is implicity a virtual method?
10. By default, access modifiers is internal, which can only be accessed in same assembly. Use protected, member can be accessed in a class that is derived from other one.
11. Async: any method using await must be marked async, await means that processing cannot continue until the task finishes. (https://stackoverflow.com/questions/14455293/how-and-when-to-use-async-and-await)
12. Lambda: use lambda expressions that requires instances of delegate types or expression trees. When call enumerable.select, use Func<T,TResult>; when call queryable.select, use tree type Expression<Func<TSource, TResult>>. One parameter, parentheses are optional, otherwise are required. Tuple type as an argument to a lambda expression.
13. Delegate is a reference type that can be used to encapsulate a named or an anonymous method. The delegate must be instantiated with a method or lambda expression that has a compatible return type and input parameters. It is a reference to a method. It used as a parameter to another method.
14. The static is not allowed in a constant declaration. Readonly vs constant, const can only be initialized at the declaration.
15. Readonly can be initialized in the declaration, in the constructor. Readonly struct(struct is immutable), ref readonly(returned reference cannot be modified)
16. Events: publisher(when) and receiver(what action), e.g. button click
17. Extern: combine with DllImport, must be declared as static
18. In: parameter is contravariant.
19. New: hide an inherited member, new vs override: new causes the the original member to become hidden, override extends the implementation for an inherited member.
20. Out: parameter is covariant.
21. Override: extend or modify abstract or virtual, or override implementation of an inherited method.
22. Sealed: prevent other classes from inheriting from it.
23. Static: static member belongs to type itself, not a specific object. Static class means all members of the class must be static.
24. Unsafe:unsafe context
25. Virtual: a method with virtual can be overridden by any class that inherits it. It likes abstract, other it cannot with static.
26. Volatile: a field can be modified by multiple threads that are executing same time.
27. Switch: generic cannot use null case.
28. Generic: maximize code reuse, type safe, performance
29. Struct: is used to encapsulate small groups of related variables, such as the coordinates of a rectangle. Cannot inherit from other struct, so member cannot be protected.
30. Foreach in: ref or ref readonly
31. Out parameter: to be passed by reference
32. Null-coalescing operator ??
33. Exception: don’t initialize variable at try block, otherwise it cannot be accessed at catch block. Use resources in try block, deal with exception in catch block, and release resources in finally block.
34. Checked and unchecked: checked is used as overflow checking for integral type arithmetic operations, if an expression contains one or more non-constant values, the compiler does not detect the overflow. Unchecked means ignore the overflow.
35. Fixed: related to garbage collection.
36. Lock: work on object and execute statement, any other thread is blocked and wait until the lock is released.
37. If parameters declare without in, ref, and out, it is called by value. Params: takes a variable number of arguments, single dimensional array, in: passed by reference and read only, it is performance optimization; out: passed by reference and must be written by called method. Ref: passed by reference and be read or written by called method, ref or in require that the variable be initialized before it is passed. If method is overloaded, cannot just use in, ref.
38. Yield return: return each element one at a time., yield is used in iterator, ienumerable, or inumerator.
39. Namespace implicitly have public access
40. Using static directive imports the members of a single class.
41. is determine the type of a value type(o is Employee e), constant, and is var varname.
42. Int? x means x is nullable.
43. New constraint: type argument must have a public parameterless constructor, and new() must be placed at the last.
44. Null is default value of reference type variable.
45. Add and remove: related to event
46. Partial method are implicitly private
47. Value is used in set:

Private int num;

public int Number

{

get{return num;}

Set{num = value;}

}

1. Group student by into
2. Equals in query keywords is not “==”,
3. D or d suffix double, F or f float, M or m decimal
4. X || y, if X is true, y is not evaluated. Same as &&; however, | and & need to evaluate both operands even if left side is true.
5. Exclusive OR: ^
6. If the left operand is int or long, >> set the high order empty bit to 0 if it is non-negative and set to 1 if it’s negative. If it is unit or ulong, high order empty bit position are set to 0.
7. If x is int or unit, only check low-order five bits of the right hand operand, if x is long or ulong, check low-order six bits of right hand operand.
8. For ==, >,<, >=,<=, if any of operand is NaN, the result is false.
9. Short-circuiting: one operand is null ,the rest operand is not executed.
10. From end operator ^: x[^1] indicates last element. X[a..b] indicates x[a] to x[b-1]
11. Is: return true or false, E is T v, returns true and assign E to variable v; E as T: convert result of an expression to a given reference, unlike cast (), as never throws an exception, only returns null.
12. Implicit conversions always succeed and never throw an exception or lose information, if it does, define it as explicit conversion. Use the cast operator () to invoke a user-defined explicit conversion.
13. Use & and \* in unsafe. & returns the address of its operand. \* is pointer indirection. -> combines \* and member access. Except &, all other operator cannot be applied into void.
14. Stackalloc allocates a block of memory on the memory.
15. String interpolation $: console.writeline($”test: {Math.PI:F2}”)
16. Delegate combination: if left operand id null, return value of other operand. Delegate removal: if right operand is not sublist of left operand, return left operand, if left is null, return null, if right is null, return left.
17. Conditional operator: A?b:c?d?e=a?b:(c?d:e)
18. X!: null forgiving operator
19. ?? returns the value of left operand if it isn’t null otherwise evaluate the right operand and returns its result. ?? doesn’t evaluate its right operand if left one is not null. ??= assign the value of right-hand operand to left one only if the left one is null. ??= can be replace is: variable ??= expression

If (variable is null)

{

Variable = expression;

}

1. =>: lambda expression and expression body.
2. :: operator: access member of an aliased namespace.
3. Default: default(int) is 0;
4. Operator overloading: public static Fraction operator.
5. Anonymous function: inline statement or expression that can be used wherever a delegate is expected. Lambda expression that has two parameters and returns no value can be converted to Action delegate type Action<T1,T2>; has one parameter and returns a value can be converted to a Fun<T,TResult>.

Action line = () => Console.WriteLine();

Func<int,int,bool> testForEquality = (x,y) => x==y;

Input type must be all explicit or all implicity.

Lambda expression and tuples: unnamed tuples(Item1,Item2,Item3…)

1. Query expression: must begin with a from and must end with a select or group.
2. Difference between out and ref: ref requires the variable be initialized before being passed, out requires calling method is required to assign a value before the method returns.
3. Hexadecimal: start from “0x”, followed by digits.

0x1A2F = (1\*163)+(10\*162)+(2\*161)+(15\*160)=6703

Each hex digit represent 4 bit value which is from 0 to 15. Values from 0-9 are represented by decimal digits. Values from 10-15 use A-F.

1. Base64: A is 000000, A-Z, a-z, 0-9, + and /. While / is 111111.

Salted Hash

Store password:1. generate a long random salt using a CSPRNG.2. concatenate the password to the salt and hash it with a hashing function 3. Save both the hash and salt to the user’s database record.

Validate password: 1. retrieve the user’s salt and hash from the DB.2.concatenate the entered password to the salt and hash it. 3, compare the hashed password in the DB with the entered one if matching, grant access.

From <https://medium.com/@mehanix/lets-talk-security-salted-password-hashing-in-c-5460be5c3aae>

In Practics, database should have 3 cloumns: username, hashedpassword, salt

public static bool VerifyPassword(string enteredPassword, string storedHash, string storedSalt){

var saltBytes = Convert.FromBase64String(storedSalt);

var rfc2898DeriveBytes = new Rfc2898DeriveBytes(enteredPassword, saltBytes, 10000);

return Convert.ToBase64String(rfc2898DeriveBytes.GetBytes(256)) == storedHash;}

public void Login\_click(object sender, EventArgs r){

//You code here

User user = GetUserByUsername(txtUsername.Text);

bool isPasswordMatched = VerifyPassword(txtpassword.Text, user.Hash, user.Salt);

if (isPasswordMatched)

{

//Login Successfull

}

else

{

//Login Failed

}

//Your code here}

<https://stackoverflow.com/questions/52146528/how-to-validate-salted-and-hashed-password-in-c-sharp>