1. OOP

Obyekt yönümlü proqramlaşdırma paradiqması (OOP) problemin həllinə köklü fərqli yanaşma aparır. OOP həll edilməli olan problemə diqqət yetirmək əvəzinə, sistemi meydana gətirən obyektlərə diqqət yetirir.

**Obyekt:**

Obyektlər, obyekt yönümlü proqramlaşdırma (OOP) dilinin əsas elementləridir. Bir obyekt həm vəziyyətə, həm də davranışa sahibdir.

Bir obyekt, adlandıra biləcəyimiz və təsvir edə biləcəyimiz bir şeydir. Obyektlər, proqramlarımızda modelləşdirmək istədiyimiz ‘şeylər’ dir və buna görə obyekt yönümlü bir proqramın məqsədi real dünyanı kodla təmsil etməkdir. Altıncı sinif kollecinin nümunəsini nəzərdən keçirsək, obyektlər tələbələr, müəllimlər, sinif otaqları və kompüterlər kimi şeylər olacaqdır.

**Class:**

Classları şablon kimi düşünsək, obyektlər şablonun tətbiqidir. Məsələn, əgər insan, bir insanın ola biləcəyi davranış və xüsusiyyətləri müəyyən edən bir classdırsa, biz də insan olmağın bütün tələblərini yerinə yetirdiyimiz üçün siz və mən bu insan classının obyektləriyik. Və ya avtomobili bir class olaraq düşünsək, müəyyən bir Honda Civic avtomobili həmin classın obyekti olardı. Bir avtomobilin sahib olduğu mühərrik, sükan, fənərlər və s.onun xüsusiyyətlərini , irəliləməsi, geriyə doğru hərəkət etməsi və s. onun davranışları olacaqdır. Obyekt yönümlü paradiqmanın gerçək dünyaya necə aid ola biləcəyini görə bilərik. Demək olar ki, real dünyada hər şeyi siniflər və obyektlər baxımından düşünmək olar, beləliklə bu hal OOP-ni populyar hala gətirir.

Obyekt və classlar anlayışları daxildə bir-birinə bağlıdır və obyekt yönümlü paradiqmanın əsasını təşkil edir.

Bir class ümumi davranış nümayiş etdirən, eyni xarakterik xüsusiyyətlərə malik olan obyektlər toplusunu təmsil edir. Ondan yarana biləcək obyektlərin planını və ya təsvirini verir. Bir class üzvü olaraq bir obyektin yaradılması instantasiya adlanır. Beləliklə, obyekt bir class nümunəsidir.

**Constructor (Konstruktor) nədir?**

Konstruktor sinif daxilində obyekt yaradılan zaman avtomatik olaraq çağırılan xüsusi metoddur. Constructor, obyekt yaradıldıqda işə düşən və obyektin atributlarını, funksiyalarını tərkibində saxlayan metoddur (initialization method).

**C# -da nümunə: (bunun icinde obyekt,class,konstruktor var hamsin burdan izah etmek olar)**

using System;

namespace Constructor {

//class yaratmaq

class Car {

string brand;

int price;

public void move() {

Console.WriteLine(“Car is moving!”);

}

// parametrli konstruktor

Car(string theBrand, int thePrice) {

brand = theBrand;

price = thePrice;

}

static void Main(string[] args) {

// parametrli konstruktorun çağırılması

Car car1 = new Car("Bugatti", 50000); //obyekt yaratmaq

Console.WriteLine("Brand: " + car1.brand);

Console.WriteLine("Price: " + car1.price);

Console.ReadLine();

}

}

}

Kodun çıxışı:

Brand: Bugatti

Price: 50000

**Destruktorlar**

Destruktor obyektin life cycle`i bitdikdə çağırılan xüsusi metoddur (bütün proqramlaşdırma dillərində mövcud deyil).

Onun məqsədi obyektlə əlaqəli resursları (məsələn, yaddaş) buraxmaqdır.

**Obyekt yönümlü Paradiqmanın 4 əsas xarakteristikası**

**Abstraction (mücərrədlik)**

Abstraksiya obyekt yönümlü proqramlaşdırma (OOP) dillərinin əsas anlayışlarından biridir. Əsas məqsədi istifadəçidən lazımsız detalları gizlətməklə mürəkkəbliyi idarə etməkdir. Bu, istifadəçiyə bütün gizli mürəkkəbliyi başa düşmədən və ya düşünmədən təqdim edilən abstraksiyanın üstündən daha mürəkkəb məntiq tətbiq etməyə imkan verir. Abstraksiya, əsas məqama yönəltmək və yersizliyi ləğv etməkdir. Abstarsiya kontextdən asılıdır. Mələsən baytar həkim pişiyə baxdıqda onun ağlına pişiyin anatomiyası, pişiyin sahibinin ağlına isə onun şirin olması rəngi və s gəlir. Həqiqi bir abstraksiya nümunəsi bir avtomobildir. Bir avtomobili idarə etmək üçün, həqiqətən, avtomobilin kapotun altında nə olduğunu bilməməyimizə gərək yoxdur.

**Encapsulation (əhatə etmə)**

Bir Class`da verilən məlumatlara digər Class`lardan əlçatanlığın idarə edilməsidir. Bu məlumatlar public, private, protected adlandırılan terminlərlə (access modifiers) idarə olunur. Public kimi təyin etdiyimiz məlumat qalan bütün siniflər üçün əlçatandır. Private kimi təyin etdiyimiz məlumat sadəcə həmin Class üçün əlçatandır, digər Class`lar bu məlumatı əldə edə bilmir. Protected ilə təyin etdiyimiz izə bu Class`dan miras alan (inherit) digər Class`lar üçün əlçatandır.

Abstraction — lazım olmayan məlumatları gizlətmək üçün istifadə olunur, Encaptulation — məlumatın gizliliyini idarə etmək üçün istifadə olunur.

**Inheritance (Miras alma)**

İnheritance (miras alma) — Müəyyən bir class`dan özü əsasında yeni class yaradılır. Yəni, əsas class`ın xarakteristikaları digər class`lara miras verilir. Belə classların 2 növü var: Child class və ya sub-class (alt sinif) və Parent class və ya super class (üst sinif). İnheritance vasitəsilə kodda yersiz təkrarın qarşısı alınır. Bənzərliklər çox vaxt dünyada mövcuddur. Yuxarıdakı şəkildəki kimi, ABŞ-ın iki prezidenti Corc H.W Buş (41-ci Prezident) ilə Corc W. Buş (43-cü Prezident) arasında bəzi oxşarlıqların olduğunu asanlıqla görmək olar.

**Polymorphism (çoxsahəli)**

Obyekt-Yönlü Proqramlaşdırmada (OYP) polimorfizm müxtəlif siniflərin obyektlərinin ümumi ana sinifin obyektləri kimi qəbul edilməsinə imkan verir. Bu, bir metod və ya funksiyanın çağırıldığı obyektə görə müxtəlif formalarda işləməsinə imkan yaradır.

**Statik Polimorfizm**

class RiyaziƏməliyyatlar {

public int Toplama(int a, int b) {

return a + b;

}

public int Toplama(int a, int b, int c) {

return a + b + c;

}

}

// İstifadə nümunəsi

class Program {

static void Main() {

RiyaziƏməliyyatlar riyaziyyat = new RiyaziƏməliyyatlar();

Console.WriteLine(riyaziyyat.Toplama(1, 2)); // Çıxış: 3

Console.WriteLine(riyaziyyat.Toplama(1, 2, 3)); // Çıxış: 6 } }

**Dinamik Polimorfizm**

class Heyvan {

public virtual void Ses() {

Console.WriteLine("Heyvan ses çıxarır");

}

}

class It : Heyvan {

public override void Ses() {

Console.WriteLine("It hürür");

}

}

class Pişik : Heyvan {

public override void Ses() {

Console.WriteLine("Pişik miyoldayır");

}

}

// İstifadə nümunəsi

class Program {

static void Main() {

Heyvan mənimİtim = new It();

Heyvan mənimPişiyim = new Pişik();

mənimİtim.Ses(); // Çıxış: "It hürür"

mənimPişiyim.Ses(); // Çıxış: "Pişik miyoldayır"

}

}

1. **MITRE Attack nədir?**

MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) kibertəhlükəsizlik sahəsində geniş şəkildə istifadə edilən məlumat bazasıdır. O, təhlükəsizlik mütəxəssislərinə müxtəlif hücum metodlarını daha yaxşı anlamağa, onlara qarşı müdafiə tədbirləri qurmağa və baş verən hadisələri analiz etməyə kömək edir.

**MITRE ATT&CK nədir?**

MITRE ATT&CK, kibertəhlükəsizlikdə istifadə edilən və hücum edənlərin müxtəlif mərhələlərdə istifadə etdikləri taktika və texnikaları təsvir edən məlumat bazasıdır. MITRE Corporation tərəfindən yaradılıb və mütəmadi olaraq yenilənir.

**Əsas Elementləri**

**Taktikalar:**

Hücumun məqsədlərini və niyyətlərini təsvir edir. Məsələn, ilkin giriş, icra, müqavimət, məlumat oğurluğu və s.

**Texnikalar:**

Bu məqsədlərə çatmaq üçün hücum edənlərin istifadə etdiyi spesifik metodlar. Məsələn, phishing, command line skripti işlətmə, credential dumping və s.

**Subtexnikalar:**

Texnikaların daha detallı formaları. Məsələn, texnika phishing ola bilər, subtexnika isə spear phishing və ya phishing attachment ola bilər.

**Necə istifadə olunur?**

**Hücum ssenarilərinin modelləşdirilməsi:**

Təşkilatlar hücum ssenarilərini modelləşdirərək mümkün zəif nöqtələri müəyyənləşdirə və müdafiə tədbirlərini buna uyğun qururlar.

**Müdafiə strategiyalarının inkişaf etdirilməsi:**

Təhlükəsizlik komandaları hücum texnikalarını anlamaqla daha güclü və təsirli müdafiə strategiyaları inkişaf etdirirlər.

**Hadisə reaksiyası və təhlil:**

Hücumların baş verdikdən sonra təhlil edilməsi və hücum edənlərin hansı texnikalardan istifadə etdiklərinin müəyyənləşdirilməsi.

**Nümunə**

Tutaq ki, bir təşkilat phishing hücumları ilə üzləşir. MITRE ATT&CK bazasından istifadə edərək, bu hücumların hansı taktika və texnikalardan istifadə etdiyini müəyyən edə bilərik və buna qarşı uyğun müdafiə tədbirləri planlaşdırarıq.

Taktika: Initial Access

Texnika: Phishing

Subtexnika: Spear Phishing Attachment

Bu məlumat əsasında təşkilat e-poçt təhlükəsizlik tədbirlərini gücləndirə və istifadəçiləri maarifləndirə bilər.

MITRE ATT&CK kibertəhlükəsizlik mütəxəssislərinə hücumları daha yaxşı anlamağa və onlara qarşı daha təsirli müdafiə tədbirləri qurmağa kömək edən güclü bir vasitədir. Təşkilatlar bu məlumat bazasından istifadə edərək təhlükəsizliklərini artırmaq və hücumlara qarşı daha yaxşı hazırlaşmaq imkanı əldə edirlər.

1. **Zərərverici programın dinamik analizi**

Bir digər üsul isə mail üzərindən istifadəçiyə zip və ya rar faylı göndərilir, istifadəçi bunu download edib run zaman isə arxa planda müəyyən zərərli proseslər baş verir. Bu proseslərə data oğurlanması və yaxud kompüterə uzaqdan erişimin yaranması missal ola bilər. Bu prosesləri izləmək üçün təhlükəsiz mühit olan Sandbox texnologiyasından istifadə olunur. Bu texnologiya sistem və ya şəbəkə üçün heç bir risk yaratmadan proqram təminatı tətbiqlərinin icra oluna və sınaqdan keçirilə biləcəyi təhlükəsiz və təcrid olunmuş mühitdir. O, mahiyyətcə istifadəçinin əsas əməliyyat sistemindən ayrı idarə olunan mühit yaradır və potensial zərərli və ya etibarsız kodun təhlükəsiz icrasına imkan verir.

**Zərərverici programın statik analizi**Tutaqki, sizin sandbox platformalarına accesiniz yoxdur və sizə zərərvericini manual analiz etmək lazım gəlir. Bu zaman siz onu run etmədən programın daxili string-lərini oxuyaraq nə etməli olduğunu anlamalısınız. Bu prosesi həyata keçirmək üçün isə pestudio program var. Biz bu programda zərərverici faylı open file edirik və bizə indicatorlar, library-lər, import-lar və stringlər haqqında məlumat verir.

**Hədəf haqqında passiv məlumat toplama prosesi**

Passiv kəşfiyyat kimi də tanınan passiv məlumatların toplanması hədəflə birbaşa əlaqə yaratmadan məlumat toplamaqdan ibarətdir. Bu yanaşma tez-tez kibertəhlükəsizlik mütəxəssisləri, etik hakerlər və təhlükə kəşfiyyatı analitikləri tərəfindən daha müdaxiləedici hərəkətlərə başlamazdan əvvəl hədəf haqqında ilkin məlumat toplamaq üçün istifadə olunur. Bunu yerinə yetirmək üçün ən məhşur tool-lar isə shodan.io, hunter.io və intelx.io websaytlarıdır. Shodan.io internetə qoşulmuş cihazları araşdırmaq və indeksləmək üçün nəzərdə tutulmuş axtarış motorudur. Veb səhifələri indeksləşdirən Google kimi ənənəvi axtarış sistemlərindən fərqli olaraq, Shodan serverlər, marşrutlaşdırıcılar, veb-kameralar, sənaye idarəetmə sistemləri və hətta IoT (Əşyaların İnterneti) cihazları kimi cihazlar haqqında məlumatları indeksləşdirir. Shodan istifadəçilərə coğrafi yer, cihaz növü, əməliyyat sistemi, açıq portlar və s. kimi müxtəlif filtrlərdən istifadə edərək xüsusi cihaz və ya xidmət növlərini axtarmağa imkan verir. Bu məlumat həm təhlükəsizlik tədqiqatçıları, həm şəbəkə administratorları, həm də hakerlər üçün faydalı ola bilər.

Hunter.io e-poçt ünvanlarının axtarışı və yoxlanılması üçün istifadə edilən veb əsaslı bir vasitədir. Bununla biz fişinq etmək istəyimiz koporativin mail ünvanlarını toplaya bilərik.

Intelx.io, geniş çeşidli onlayn məlumat mənbələrinə girişi təmin edən və indeksləşdirmədə ixtisaslaşan hərtərəfli axtarış mühərriki və məlumat mədən platformasıdır. O, adətən kibertəhlükəsizlik mütəxəssisləri, tədqiqatçılar, hüquq-mühafizə orqanları və bizneslər tərəfindən kəşfiyyat məlumatlarının toplanması, araşdırmaların aparılması və onlayn təhlükələrin monitorinqi üçün istifadə olunur.

1. **Data encyription**

**Şifrələmə** məlumatın şifrəli mətnə çevrildiyi məlumat təhlükəsizliyi formasıdır. Yalnız açarı olan səlahiyyətli şəxslər kodu deşifrə edə və orijinal açıq mətni(plaintext) əldə edə bilərlər. Daha sadə dillə desək, şifrələmə məlumatları icazəsiz şəxs üçün oxunmaz hala gətirmək üsuludur. Şifrələmə yalnız məlumatların və ya mesajların məxfiliyini təmin etmir, həmçinin məlumatların və ya mesajların orijinal vəziyyətindən heç bir şəkildə dəyişdirilmədiyini sübut edən autentifikasiya və tamlığı təmin edir.

Məlumatı şifrələmək üçün siz şifrələmə alqoritmindən(cipher) istifadə edilir. Şifrələmə alqoritmini həmçinin şifrələmə açarı da adlandırmaq olar.

**Qədim şifrələmə metodları**

Şifrələmənin kökləri minlərlə il əvvələ qayıdır. Qədimdə insanlar mesajları yad gözlərdən gizlətmək üçün müxtəlif üsullardan istifadə edirdilər. Şifrələmənin ən qədim məlum nümunələrindən biri, hərbi səfərlər zamanı generallara göndərilən mesajları şifrələmək üçün istifadə edilən Yuli Sezarın adını daşıyan Sezar şifrələmə üsuludur. Bu üsul əlifbanın hər hərfini Sezar sürüşməsi kimi tanınan müəyyən sayda mövqe ilə dəyişməyi nəzərdə tuturdu. Müasir standartlara görə sadə olsa da, Sezar şifrəsi gələcəkdə daha kompleks şifrələmə metodlarının əsasını qoydu. Məsələn:

Sumqayıt >>>>> Uwoscakv (n=2)

Bu metoddan istifadə etdikdə gizli mesaj düşmənin əlinə keçsə belə düşmən ondan yararlana bilməyəcəkdi.

Qədim şifrələmə metodlarına digər nümunələr aşağıdakılardır:

• Əvəzetmə Şifrələri – Bu şifrələr sabit bir sistem olan alqoritmdən istifadə edərək düz mətni şifrəli mətnlə əvəz edir. Açar sabit sistemi göstərən sənəddir və şifrələməni tərsinə çevirmək üçün istifadə olunur.

• Transpozisiya Şifrələri – Bu alqoritm mətnin sırasını daha sonra şifrələnə bilən fərqli permutasiyalara dəyişdirmək üçün açar kimi xidmət edən bir sıra qaydalardan istifadə edir.

• Polialfabetik Şifrələr – Bunlar şifrəli mətnin icazəsiz deşifrəsini daha da çətinləşdirmək üçün çoxlu əlifbalardan istifadə edən əvəzetmə şifrənin növüdür.

• Nomenklator şifrələri – Ümumi açıq mətn sözlərini simvollarla əvəz edən əvəzetmə şifrələmə növüdür və bu şifrələmə metodu şifrə analizi(cryptanalysisin) müəyyən növünün üstəsindən gəlməyə çalışır.

**Kriptanaliz** kriptosistemlərin zəif tərəflərini tapmaq məqsədi ilə onların öyrənilməsidir. Təxminən eramızın 800-cü illərində yaşamış ərəb riyaziyyatçısı Əl-Kindiyə aid olan kriptoanalizin ən geniş yayılmış formalarından biri tezlik analizi adlanır. O, təkrarlanan simvollar üçün şifrəli mətni araşdırır və şifrəsi açılan mesajda yüksək tezlikdə görünəcək sözlərlə çarpaz istinad edir.

**Müasir şifrələmə**

Müasir şifrələmə metodları haqqında danışmazdan əvvəl açıq(public) və gizli(private) açarlar və rəqəmsal inqilabın şifrələməni necə dəyişdiyi haqqında bir az danışmalıyıq. İndiyə qədər nəzərdən keçirdiyimiz bütün nümunələr Gizli Açar Kriptoqrafiyası adlandırdığımız nümunələrdir. O vaxtlar, şifrələmə tamamilə gizli açardan asılı idi və şifrənin açılması üçün açarın fiziki şəkildə ötürülməsi lazım idi. Gizli açarınızın oğurlanmasının dəhşətli nəticələri ola bilərdi. Onu fiziki olaraq daşımaq və ötürmək məcburiyyətində olmaq bu riski daha da artırırdı.

Bu gün kompüter texnologiyası və internet sayəsində biz Açıq Açar Kriptoqrafiyasını tətbiq edə bilərik. Açıq açar kriptoqrafiyasında, bir açıq açar məlumatı şifrələmək üçün, digər gizli açar isə şifrəni açmaq üçün istifadə olunur. Bunu SSL handshake zamanı müşahidə edə bilərik.

Bu gün şifrələmənin ən geniş yayılmış formaları bunlardır:

• Asimmetrik Şifrələmə – Açıq açar kriptoqrafiyası kimi də tanınan asimmetrik şifrələmə, məlumatları şifrələmək və deşifrə etmək üçün iki fərqli, lakin əlaqəli açardan istifadə edən nisbətən yeni bir üsuldur. Bu açarlardan biri gizli açar, biri isə açıq açardır. Açıq açar məlumatı şifrələmək üçün, gizli açar isə şifrəni açmaq üçün istifadə olunur. Açıq açarın təhlükəsizliyinə ehtiyac yoxdur, çünki o, hamı üçün əlçatandır və internet üzərindən paylaşıla bilər.

• Simmetrik şifrələmə - Bu sadə şifrələmə metodunda məlumatı həm şifrələmək, həm də deşifrə etmək üçün yalnız bir açardan istifadə edilir. Ən qədim və ən məşhur şifrələmə metodu olsa da, onun ən əsas çatışmamazlığı ondan ibarətdir ki, məlumatı deşifrələməmişdən qabaq hər iki tərəfdə də açar olmalıdır.

**Müasir şifrələmə alqoritmləri**

• Data Encryption Standard (DES) - məlumat şifrələməsinin köhnəlmiş simmetrik açar üsuludur. DES mesajı şifrələmək və deşifrə etmək üçün eyni açardan istifadə etməklə işləyir, ona görə də həm göndərən, həm də qəbul edən eyni şəxsi açara sahib olmalıdır. DES daha təhlükəsiz AES alqoritmi ilə əvəz edilmişdir. O, 1977-ci ildə ABŞ hökuməti tərəfindən dövlət məlumatlarının şifrələnməsi üçün rəsmi standart kimi qəbul edilib. DES müasir kriptoqrafiya və şifrələmə sənayesi üçün təkan olmuşdur.

• Triple Data Encryption Standard (3DES) - DES alqoritmini üç ayrı açarla üç dəfə işlətməyi nəzərdə tuturdu. Tək DES alqoritmi getdikcə bruteforce hücumlarına qarşı durmaq üçün çox zəif hesab olunurdu və daha güclü AES hələ də qiymətləndirmə mərhələsində idi.

• RSA - Rivest-Şamir-Adleman (RSA) alqoritm və kriptosistemin əsasını təşkil edir - xüsusi təhlükəsizlik xidmətləri və ya məqsədləri üçün istifadə olunan kriptoqrafik alqoritmlər dəstidir. Bu alqoritmlər tez-tez brauzerlər tərəfindən vebsaytlara və virtual şəxsi şəbəkələrə (VPN) qoşulmaq üçün istifadə olunur. RSA asimmetrikdir, burada şifrələmə üçün iki fərqli açar istifadə olunur: biri açıq və biri gizli. Şifrənin açılması açıq açarla aparılırsa, şifrələmə gizli açarla və ya əksinə həyata keçirilir.

• Advanced Encryption Standard (AES) – DES alqoritminə alternativ olaraq Milli Standartlar və Texnologiya İnstitutu (NIST) tərəfindən 1997-ci ildə hazırlanmış AES, ABŞ hökuməti tərəfindən həssas məlumatları qorumaq üçün seçilmişdir.

**Bulud şifrələməsi**

Bulud şifrələməsi bulud saxlama provayderləri tərəfindən təklif olunan bir xidmətdir ki, burada verilənlər buluda ötürülməzdən əvvəl ilk olaraq alqoritmlərdən istifadə edərək şifrələnir. Bulud saxlama provayderinin müştəriləri provayderin şifrələmə və şifrələmə açarlarının idarə edilməsi üzrə siyasət və prosedurlarının dərinlik səviyyəsindən xəbərdar olmalı və razı olmalıdır.

Şifrələmə daha çox bandwidth istifadə etdiyinə görə, bir çox bulud provayderi yalnız parollar və hesab nömrələri kimi bir neçə verilənlər bazası sahəsində sadə şifrələmə təklif edir. Bu, çox vaxt bəzi təşkilatlar üçün yetərsizdir. Beləliklə onlar istədikləri təhlükəsizlik səviyyəsinə çatmaq üçün öz şifrələmə proqramlarından istifadə etdikləri və öz şifrələmə açarlarını idarə etdikləri BYOE( Öz Şifrələməni Gətir) modelinə etibar edirlər.

**End-to-End encryption**

End-to-end encryption (E2EE) yalnız bir-biri ilə əlaqə saxlayan iki istifadəçinin mesajları oxuya bilməsini təmin edir. Hətta telekommunikasiya və ya internet provayderi kimi vasitəçilər də mesajların şifrəsini aça bilmir. E2EE ümumiyyətlə gizli və təhlükəsiz onlayn ünsiyyət qurmağın ən təhlükəsiz yolu kimi qəbul edilir. İstifadə olunan E2EE nümunələri arasında istifadəçilərin mesajlarının "kilidlərlə" qorunduğunu təsdiqləyən WhatsApp mesajlaşma xidməti daxildir.

1. **Təhlükəsiz Şəbəkə Protokolları**

Şəbəkə protokolu müxtəlif qurğular arasında məlumatların necə ötürüldüyünü müəyyən edən müəyyən edilmiş qaydalar toplusudur. Əsasən, daxili proseslərində, strukturunda və ya dizaynında hər hansı fərqdən asılı olmayaraq cihazların bir-biri ilə əlaqə saxlamasına imkan verir. Şəbəkə protokolları müasir rəqəmsal rabitədə mühüm rol oynayır və bütün dünyada insanlarla asanlıqla ünsiyyət qurmağımıza səbəb olur.

Şəbəkə protokollarının necə işlədiyini başa düşmək, bir-biri ilə əlaqələndirilmiş cihazların şəbəkə üzərindən necə ünsiyyət qurduğunu görmək çox vacibdir. Ən populyar model, OSI kompüter sistemlərinin şəbəkə üzərindən necə əlaqə saxladığını izah edir. Bu yeddi qatlı model yeddi qat üzrə iki şəbəkə cihazı arasında əlaqə prosesini vizuallaşdırır.

Şəbəkə protokolları rabitə prosesini hər bir OSI model təbəqəsi üzrə diskret tapşırıqlara bölür. Şəbəkə əlaqəsini aktivləşdirmək üçün hər bir təbəqədə bir və ya bir neçə protokol işləyir. Məsələn, İnternet Protokolu (IP) şəbəkədən şəbəkəyə rabitəni təmin etmək üçün məlumat paketlərinin mənbə ünvanı və təyinatı kimi məlumatları idarə edərək məlumatları yönləndirir. Buna görə də ona şəbəkə qatı protokolu da deyilir.

OSI modelinin səviyyələri:

1. Fiziki səviyyə(Physical layer)
2. Kanal səviyyəsi(Data Link layer)
3. Şəbəkə səviyyəsi( Network layer)
4. Nəqliyyat səviyyəsi (Transport layer)
5. Sessiya səviyyəsi ( Session layer)
6. Təqdimat səviyyəsi (Presentation layer)
7. Tətbiq səviyyəsi (Application layer)

Təhlükəsiz şəbəkə protokolları kibertəhlükəsizliyin əsasını təşkil edir. Onlar şəbəkələrdə məlumatların bütövlüyünü, məxfiliyini və əlçatanlığını qorumaq üçün nəzərdə tutulmuşdur. Kiber təhlükələr inkişaf etdikcə bu protokollar da inkişaf edir.

* **Nəqliyyat Səviyyəsi Təhlükəsizliyi (TLS) 1.3**: Secure Sockets Layer və Transport Layer Security (SSL/TLS) OSI modelinin Səviyyə 5-də işləyən şəbəkə təhlükəsizliyi protokoludur. Bu protokol məlumatların şifrələnməsi, autentifikasiyası və bütövlüyün qorunması da daxil olmaqla, şəbəkə trafiki üçün bir sıra üstünlüklər təklif edir.
* **Datagram Transport Layer Security (DTLS**): SSL/TLS sabit, etibarlı bağlantılar yaradan TCP ilə işləmək üçün nəzərdə tutulmuşdur. Bununla belə, İstifadəçi Datagram Protokolu (UDP) əlaqəsiz ötürmə təklif edən başqa və çox istifadə olunan protokoldur.

Datagram Nəqliyyat Səviyyəsi Təhlükəsizliyi (DTLS) protokolu OSI modelinin Səviyyə 5-də işləyir və SSL/TLS-dən əldə edilib, lakin əlaqəsiz, etibarsız dataqramlarla istifadə üçün nəzərdə tutulub. Bu protokol video konfrans, VoIP və onlayn oyun daxil olmaqla, aşağı gecikmə tələb edən proqramlar üçün məlumatların bütövlüyünü və məxfiliyini təmin etməyə kömək edir.

* **Kerberos** xidmət sorğularının autentifikasiyası üçün geniş istifadə olunan Səviyyə 7 protokoludur. O, etibarsız ictimai şəbəkələr üzərindən etibarlı sistemlər arasında sorğuların autentifikasiyası üçün nəzərdə tutulmuşdur.

Kerberos şəxsiyyət və autentifikasiya sübutunu təmin etmək üçün istifadə edilən biletlər konsepsiyası ətrafında qurulub. İstifadəçinin şəxsiyyətini yoxlamaq və bu nişanları yaratmaq üçün mərkəzləşdirilmiş identifikasiya serverindən istifadə olunur. Bundan sonra, bu serverə güvənən hər bir sistem və ya proqram istifadəçinin müəyyən sorğu vermək hüququnu yoxlaya bilər.

* **SNMPv3** Sadə Şəbəkə İdarəetmə Protokolu (SNMP) təşkilatın cihazlarını izləmək və idarə etmək üçün Səviyyə 7 şəbəkə protokoludur. SNMP meneceri cihazları sorğulaya bilər və cihazdakı agent bu sorğulara cavab verir və ya tələb olunan tapşırıqları yerinə yetirir. SNMP-nin ilk versiyaları etibarsız idi, şifrələmə, autentifikasiya və bütövlüyün qorunmasından məhrum idi. SNMPv3 müasir, təhlükəsiz kriptoqrafik alqoritmlərdən istifadə edərək bütün bu xüsusiyyətləri təklif edən 2004-cü ildə təqdim edilmiş protokolun yenilənmiş versiyasıdır.
* **HTTPS:** HTTP “web browsing” üçün istifadə olunan Səviyyə 7 şəbəkə protokoludur. HTTP ən erkən şəbəkə protokollarından biridir və şifrələnməmiş, insan tərəfindən oxuna bilən protokol kimi həyata keçirilmişdir. Şəbəkə trafikini dinləyən hər kəs təyinat yerinə gedən yolda HTTP trafikini oxumaq və dəyişdirmək imkanına malikdir. Ödəniş kartı məlumatı və ya login məlumatları kimi həssas məlumatların ötürülməsi üçün İnternet getdikcə daha çox istifadə edildikdə bu, əhəmiyyətli təhlükəsizlik riski yaradırdı. HTTPS verilənlərin şifrələnməsi, bütövlüyün qorunması və autentifikasiyası təklif edən HTTP protokolunun təhlükəsiz versiyasıdır. Protokolu sıfırdan yenidən yazmaq əvəzinə, HTTPS SSL/TLS paketi daxilində HTTP işlətməklə həyata keçirildi.
* **Virtual özəl şəbəkələr (VPN**) OSI modelinin müxtəlif səviyyələrində işləyə bilər. Məsələn, VPN protokolu olan IPsec 3-cü səviyyədə işləyir. Bununla belə, səviyyə 7 protokolu olan HTTPS-dən istifadə edərək VPN-ləri tətbiq etmək də mümkündür.

VPN-lər iki paradiqmadan birində istifadə edilə bilər. Uzaqdan giriş VPN-ləri istifadəçinin kompüterini korporativ şəbəkədə yerləşən uzaq serverə qoşmaq üçün nəzərdə tutulub. Digər tərəfdən site-to-site VPN-lər, etibarsız ictimai İnternet üzərindən iki coğrafi paylanmış şəbəkəni birləşdirmək üçün nəzərdə tutulmuşdur.

* **SSH:** Secure Shell (SSH) protokolu təhlükəsiz olmayan şəbəkə üzərindən kompüterə əmrlərin təhlükəsiz şəkildə göndərilməsi üsuludur. SSH cihazlar arasında əlaqələri autentifikasiya etmək və şifrələmək üçün kriptoqrafiyadan istifadə edir.
* **SFTP** (SSH File Transfer Protocol) təhlükəsiz fayl ötürmə protokoludur. SSH protokolu üzərində işləyir. SSH-nin tam təhlükəsizlik və autentifikasiya funksiyasını dəstəkləyir. SFTP, fayl ötürmə protokolu kimi köhnə FTP-ni demək olar ki, əvəz etmişdir. O, bu protokolların təklif etdiyi bütün funksionallığı özündə ehtiva edir, lakin daha təhlükəsiz və daha etibarlı, daha asan konfiqurasiya təklif edir.

1. **Massivlər və siyahılar, kolleksiyalar vasitəsilə iterasiya, kolleksiyalardan istifadə edən proqramların yaradılması. Kolleksiyalar haqqında daha çox məlumat: Dictionary, List. Əsas LINQ sorğuları.**

Massiv oxşar tipli verilənlər toplusudur.

// declare an array

int[] age;

// allocate memory for array

age = new int[5];

//declare and allocate

int[] age = new int[5];

C#-da biz massivi elan edən zaman onu inisiallaşdıra bilərik. Misal üçün,

int [] numbers = {1, 2, 3, 4, 5};

Massivdə hər bir massiv elementinin mövqeyini təyin etmək üçün indeks nömrəsindən istifadə edirik.

* Massiv indeksi həmişə 0-dan başlayır. Yəni massivin birinci elementi 0 indeksindədir.

Numbers[2] ---- 3

1. ***Dövrlərdən istifadə edərək massivin elementlərinin təkrarlanması***

C#-da massivin hər bir elementini təkrarlamaq üçün dövrlərdən istifadə edə bilərik. Misal üçün, for dövründən istifadə qaydasına baxaq:

static void Main(string[] args) {

int[] numbers = { 1, 2, 3};

for(int i=0; i < numbers.Length; i++) {

Console.WriteLine("Element in index " + i + ": " + numbers[i]);

}

Console.ReadLine();

}

Foreach ile:

static void Main(string[] args) {

int[] numbers = {1, 2, 3};

Console.WriteLine("Array Elements: ");

foreach(int num in numbers) {

Console.WriteLine(num);

}

Console.ReadLine();

}

Çoxölçülü massivdə massivin hər bir elementi həm də massivdir. Misal üçün,

int[ , ] x = { { 1, 2 ,3}, { 3, 4, 5 } };

Burada ***x*** iki elementi olan çoxölçülü massivdir: **{1, 2, 3}** və **{3, 4, 5}**. Və massivin hər bir elementi də 3 elementdən ibarət massivdir.

* **İkiölçülü massiv**

İki ölçülü massiv onun elementləri kimi birölçülü massivlərdən ibarətdir. O, müəyyən sayda sətir və sütundan ibarət bir cədvəl kimi təqdim edilə bilər.

int[ , ] x = new int [2, 3];

C#-da biz massivi elan edən zamanı massivi inisiallaşdıra bilərik. Misal üçün,

int[ , ] x = { { 1, 2 ,3}, { 3, 4, 5 } };

İkiölçülümassivin elementlərinə daxil olmaq üçün indeks nömrəsindən istifadə edirik. Misal üçün,

// a 2D array

int[ , ] x = { { 1, 2 ,3}, { 3, 4, 5 } };

// access first element from first row

x[0, 0]; // returns 1

İkiölçülü massivin elementlərini də dəyişə bilərik. Elementi dəyişdirmək üçün biz sadəcə həmin indeksə yeni qiymət təyin edirik. Misal üçün,

static void Main(string[] args) {

int[ , ] numbers = {{2, 3}, {4, 5}};

// old element

Console.WriteLine("Old element at index [0, 0] : "+numbers[0, 0]);

// assigning new value

numbers[0, 0] = 222;

// new element

Console.WriteLine("New element at index [0, 0] : "+numbers[0, 0]);

}

* ***Dövrdən istifadə edərək ikiölçülü massivin elementlərinin təkrarlanması***
* static void Main(string[] args) {
* int[ , ] numbers = { {2, 3, 9}, {4, 5, 9} };
* for(int i = 0; i < numbers.GetLength(0); i++) {
* Console.Write("Row "+ i+": ");
* for(int j = 0; j < numbers.GetLength(1); j++) {
* Console.Write(numbers[i, j]+" ");
* }
* Console.WriteLine();
* }
* **numbers.GetLength(0)** - ikiölçülü massivdə sətirlərin sayını verir.
* **numbers.GetLength(1)** - sıradakı elementlərin sayını verir.
* **C# Proqramlaşdırma Dilində Jagged Massivlər**

C#-da jagged massiv çoxsaylı massivlərdən ibarətdir. Bununla belə, çoxölçülü massivlərdən fərqli olaraq, jagged massivin içindəki hər massiv müxtəlif ölçülərdə ola bilər.

static void Main(string[] args) {

// create a jagged array

int[ ][ ] jaggedArray = {

new int[] {1, 3, 5},

new int[] {2, 4},

};

// print elements of jagged array

Console.WriteLine("jaggedArray[1][0]: " + jaggedArray[1][0]);

Console.WriteLine("jaggedArray[1][1]: " + jaggedArray[1][1]);

Console.WriteLine("jaggedArray[0][2]: " + jaggedArray[0][2]);

Console.ReadLine();

}

**C# Proqramlaşdırma Dilində Kolleksiyalar**

C# proqramlaşdırma dilində kolleksiyalar ənənəvi massivlərdən daha böyük imkanlar və funksionallıqlar təmin edən System.Collections namespace-də mövcud olan əvvəlcədən təyin edilmiş siniflər toplusudur. C#-dakı kolleksiyalar təkrar istifadə edilə bilən, daha güclü və daha səmərəlidir və ən əsası keyfiyyət və performansı təmin etmək üçün dizayn edilmiş və sınaqdan keçirilmişdir.

Beləliklə, sadə sözlərlə, C#-da kolleksiyanın dinamik massiv olduğunu deyə bilərik. Bu o deməkdir ki, C#-da kolleksiyalar çoxlu dəyərləri saxlamaq imkanına malikdir, lakin aşağıdakı xüsusiyyətlərə malikdir.

* Ölçüsü dinamik şəkildə artırıla bilər.
* Kolleksiyanın ortasına element daxil edə bilərik.
* O, həmçinin kolleksiyanın ortasından elementləri silmək imkanı verir.

C#-dakı kolleksiyalar bir qrup obyekti təmsil edən siniflərdir. C# Collections proqramının köməyi ilə biz obyektlər üzərində yaddaşda saxlamaq, yeniləmək, silmək, axtarmaq və sıralamaq və s. kimi müxtəlif növ əməliyyatları yerinə yetirə bilərik. C# proqramlaşdırma dilində kolleksiyalar 3 ümumi namespace-ə bölünür:

* ***System.Collections***

C# dilində qeyri-ümumi kolleksiya System.Collections namespace-də müəyyən edilmişdir. O, obyekt istinadları üzərində işləyən ümumi təyinatlı verilən strukturudur, ona görə də istənilən növ obyekti idarə edə bilər, lakin safe-type deyil. Qeyri-ümumi kolleksiyalar interfeyslər və siniflər dəsti ilə müəyyən edilir. Aşağıdakı cədvəl, System.Collections namespace-də ən çox istifadə olunan sinifləri göstərir:

|  |  |
| --- | --- |
| **Class name** | **Description** |
| [**ArrayList**](https://www.geeksforgeeks.org/c-arraylist-class/) | Dinamik massivdir, yəni, massivin ölçüsü sabit deyil, run time-da artıb azala bilər. |
| [**Hashtable**](https://www.geeksforgeeks.org/c-hashtable-class/) | Bu, açarın hash kodu əsasında təşkil edilən açar və dəyər cütlərinin toplusunu təmsil edir. |
| **Queue** | FİFO prinsipiylə təşkil olunmuş kolleksiyanı təmsil edir. |
| **Stack** | Bu xətti məlumat strukturudur. Elementlərin daxil edilməsi və silinməsi LİFO nümunəsinə uyğundur. |

* ***System.Collections.Concurrent***

System.Collections.Concurrent thread-safe kod əldə etməyə kömək edən bəzi kolleksiya sinifləri təqdim edir.

**Thread safety nədir?**

Çoxlu thread-lərin eyni kod parçasını icra etməyə çalışdığı vəziyyətlər ola bilər. Kod eyni vaxtda daxil olan birdən çox thread-dən asılı olmayaraq düzgün icra oluna bilsə, "thread-sfae" olduğu deyilir.

Kolleksiyaya çoxsaylı thread-lər daxil olduqda System.Collections və System.Collections.Generic siniflərindən daha çox System.Collections.Concurrent siniflərindən istifadə etmək tövsiyə olunur.

|  |  |
| --- | --- |
| **Class name** | **Description** |
| **BlockingCollection** | IPoducerConsumerCollection tətbiq edən thread-safe kolleksiyalar üçün bloklama və məhdudlaşdırma imkanlarını təmin edir. |
| **ConcurrentBag** | Obyektlərin sırasız thread-safe toplusunu təmsil edir. |
| **ConcurrentDictionary** | Açar/dəyər cütlərinin thread-safe kolleksiyasını təmsil edir. |
| **ConcurrentQueue** | Thread-safe dəstəkli FİFO kolleksiyasını təmsil edir. |
| **ConcurrentStack** | Thread-safe dəstəkli LİFO kolleksiyasını təmsil edir. |

* ***System.Collections.Generic***

C# dilində generic kolleksiyalar, System.Collections.Generic namespace-də təyin olunur. Bu, linkedlist, stack, queue və dictionary kimi standart məlumat strukturlarının ümumi tətbiqini təmin edir. Bu kolleksiyalar safe-type-dır, çünki onlar generic-dir, yəni yalnız kolleksiyanın tipi ilə uyğun tipdə olan obyektlər generic kolleksiyada saxlanıla bilər, bu da təsadüfi tip uyğunsuzluqlarını aradan qaldırır. Generic kolleksiyalar, interfeys və siniflər dəsti ilə təyin edilir.

Aşağıdakı cədvəl, System.Collections.Generic namespace-də ən çox istifadə olunan sinifləri göstərir:

|  |  |
| --- | --- |
| **Class name** | **Description** |
| **Dictionary<TKey,TValue>** | O, açar/dəyər cütlərini saxlayır və non-generic Hashtable sinifində tapılanlara bənzər funksionallığı təmin edir. |
| [**List<T>**](https://www.geeksforgeeks.org/c-list-class/) | Bu non-generic ArrayList sinifində tapılana bənzər funksionallığı təmin edən type-safe dinamik massivdir. |
| **Queue<T>** | Bu, FIFO prinsipiylə təşkil olunmuş siyahıdır və non-generic Queue sinfində tapılanlara bənzər funksionallığı təmin edir. |
| **SortedList<TKey,TValue>** | Bu, açar/dəyər cütlərinin çeşidlənmiş siyahısıdır və non-generic SortedList sinifində tapılanlara oxşar funksionallığı təmin edir. |
| **Stack<T>** | Bu, LIFO prinsipiylə təşkil olunmuş siyahıdır və non-generic Stack sinfində tapılanlara bənzər funksionallığı təmin edir. |
| [**HashSet<T>**](https://www.geeksforgeeks.org/c-sharp-hashset-class/) | Bu unikal elementlərin nizamsız toplusudur. Kolleksiyaya dublikatların daxil edilməsinin qarşısını alır. |
| [**LinkedList<T>**](https://www.geeksforgeeks.org/c-sharp-linkedlist-class/) | Elementlərin sürətli daxil edilməsinə və çıxarılmasına imkan verir. Klassik linkedlisti implement edir. |

**C# proqramlaşdırma dilində List<T>**

List<T>, indekslə daxil oluna bilən və siyahının çeşidlənməsi, axtarışı və dəyişdirilməsi üsullarına malik eyni tipli obyektlərin toplusudur. Bu, ArrayList-in System.Collections.Generic namespace-i altında gələn generic versiyasıdır.

List<T> ümumi kolleksiyadır, ona görə də onun saxlaya biləcəyi məlumat növü üçün tip parametrini təyin etməlisiniz.

public static void Main()

{

// create a list named number that contains integer data items

List<int> number = new List<int>();

number.Add(1);

number.Add(2);

Console.WriteLine(number[1]); //

}

Siyahıya indeks, for/foreach dövrləri və LINQ sorğularından istifadə etməklə daxil olmaq olar:

public static void Main()

{

// create a list

List<string> languages = new List<string>() { "Python", "Java" };

// access the first and second elements of languages list

Console.WriteLine("The first element of the list is " + languages[0]);

Console.WriteLine("The second element of the list is " + languages[1]);

}

public static void Main()

{

// create a list

List<string> albums = new List<string>() { "Red", "Midnight", "Reputation" };

// iterate through the albums list

for (int i = 0; i < albums.Count; i++)

Console.WriteLine(albums[i]);

}

* **List<T> metodları**

|  |  |
| --- | --- |
| **Metod** | **İstifadəsi** |
| **Add** | Listin sonuna element əlavə edir. |
| **AddRange** | Listin sonuna kolleksiya əlavə edir. |
| **BinarySearch** | Verilmiş elementi List-də axtarır və onun indeksini geri qaytarır. |
| **Clear** | List-dən bütün elementləri silir. |
| **Contains** | Verilmiş elementin List-də mövzud olub olmadığını yoxlayır. |
| **Find** | Göstərilən predikat funksiyasına əsasən birinci elementi tapır. |
| **Foreach** | Listi iterasiya edir. |
| **İnsert** | Listdə göstərilən indeksə element daxil edir. |
| **İnsertRange** | Listdə göstərilən indeksə kolleksiya daxil edir. |
| **Remove** | Verilmiş elementin ilk tapıldığı indeksdən silir. |
| **RemoveAt** | Listdə göstərilən indeksdəki elementi silir. |
| **RemoveRange** | Göstərilən predikat funksiyasına uyğun gələn bütün elementləri silir. |
| **Sort** | Listin elementlərini sıralayır. |

**C# proqramlaşdırma dilində Dictionary<TKey,TValue>**

Dictionary<TKey, TValue> sıralanmamış açar/dəyər cütləri kimi elementlərdən ibarət ümumi kolleksiyadır. Misal üçün,

public static void Main()

{

// create a dictionary

Dictionary<int, string> country = new Dictionary<int, string>();

// add items to dictionary

country.Add(5, "Brazil");

country.Add(3, "China");

country.Add(4, "Usa");

// print value having key is 3

Console.WriteLine("Value having key 3: " + country[3]);

}

Dictionary-nin açarlarından istifadə edərək onun içindəki elementlərə daxil ola bilərik. Misal üçün,

public static void Main()

{

// create a dictionary

Dictionary<string, string> student = new Dictionary<string, string>();

// add items to dictionary

student.Add("Name", "Susan");

student.Add("Faculty", "History");

// access the value having key "Name"

Console.WriteLine(student["Name"]);

// access the value having key "Faculty"

Console.WriteLine(student["Faculty"]);

}

C#-da biz həmçinin foreach dövründən istifadə edərək dictionary-nin hər bir elementi arasında dövr edə bilərik. Misal üçün,

public static void Main()

{

// create a dictionary

Dictionary<string, string> car = new Dictionary<string, string>();

// add items to dictionary

car.Add("Model", "Hyundai");

car.Add("Price", "36K");

// iterate through the car dictionary

foreach (KeyValuePair<string, string> items in car)

{

Console.WriteLine("{0} : {1}", items.Key, items.Value);

}

}

Dictionary-dəki elementlərin dəyərini aşağıdakı kimi dəyişə bilərik:

public static void Main()

{

// create a dictionary

Dictionary<string, string> car = new Dictionary<string, string>();

// add items to dictionary

car.Add("Model", "Hyundai");

car.Add("Price", "36K");

// print the original value

Console.WriteLine("Value of Model before changing: " + car["Model"]);

// change the value of "Model" key to "Maruti"

car["Model"] = "Maruti";

// print new updated value of "Model"

Console.WriteLine("Value of Model after changing: " + car["Model"]);

}

Dictionary-dəki elementləri silmək üçün Remove() metodundan istifadə edirik:

Misal üçün,

public static void Main()

{

// create a dictionary

Dictionary<string, string> employee = new Dictionary<string, string>();

// add items to dictionary

employee.Add("Name", "Marry");

employee.Add("Role", "Manager");

employee.Add("Address", "California");

Console.WriteLine("Original Dictionary :");

// iterate through the modified dictionary

foreach (KeyValuePair<string, string> items in employee)

{

Console.WriteLine("{0} : {1}", items.Key, items.Value);

}

// remove value with key "Role"

employee.Remove("Role");

Console.WriteLine("\nModified Dictionary :");

// iterate through the modified dictionary

foreach (KeyValuePair<string, string> items in employee)

{

Console.WriteLine("{0} : {1}", items.Key, items.Value);

}

**C# proqramlaşdırma dilində LINQ sorğuları**

* ***LINQ nədir?***

Language-Integrated Query (LINQ) sorğu imkanlarının birbaşa C# dilinə inteqrasiyasına əsaslanan güclü texnologiyalar toplusudur. LINQ sorğuları siniflər, metodlar, eventlər kimi C# .NET-də birinci dərəcəli dil quruluşudur. LINQ sorğu obyektləri (LINQ to Objects), əlaqəli verilənlər bazaları (LINQ to SQL) və XML (LINQ to XML) üçün ardıcıl sorğu təcrübəsi təqdim edir.

LINQ (Language Integrated Query) müxtəlif mənbələrdən və formatlardan məlumatları əldə etmək üçün C# və VB.NET dillərində vahid sorğu sintaksisidir. O, C# və ya VB dillərində inteqrasiya olunub, bununla da proqramlaşdırma dilləri və verilənlər bazası arasında uyğunsuzluğu aradan qaldırır, həmçinin müxtəlif növ məlumat mənbələri üçün vahid sorğu interfeysi təmin edir.

Məsələn, SQL verilənlər bazasından məlumatları saxlamaq və əldə etmək üçün istifadə edilən Strukturlaşdırılmış Sorğu Dilidir. Eyni şəkildə, LINQ kolleksiyalar, ADO.Net DataSet, XML Sənədləri, veb xidməti və MS SQL Server və digər verilənlər bazaları kimi müxtəlif növ məlumat mənbələrindən məlumatları əldə etmək üçün C# və VB.NET-də qurulmuş strukturlaşdırılmış sorğu sintaksisidir.

***Sorğu sintaksisinə nümunə:***

**class** GFG {

    // Main Method

**static** **public** **void** Main()

    {

        // Creating data source

**string**[] language = {"C#", "VB", "Java", "C++",

                        "C", "Perl", "Ruby", "Python"};

        // Creating a query to get the

        // value from the data source

**var** result = **from** lang **in** language

**where** lang.Contains('C')

**select** lang;

        // display the result of the query

**foreach**(**var** l **in** result)

        {

            Console.WriteLine(l);

        }

    }

***Metod sintaksisinə uyğun nümunə:***

**class** GFG {

    // Main Method

**static** **public** **void** Main()

    {

        // Data source

        List<**string**> my\_list = **new** List<**string**>() {

                "This is my Dog",

                "Name of my Dog is Robin",

                "This is my Cat",

                "Name of the cat is Mewmew"

        };

        // Creating LINQ Query

        // Using Method syntax

**var** res = my\_list.Where(a => a.Contains("Dog"));

        // Executing LINQ Query

**foreach**(**var** q **in** res)

        {

            Console.WriteLine(q);

        }

* ***LINQ-da standart sorğu operatorları***

LINQ-da Standart Sorğu Operatorları əslində IEnumerable<T> və IQueryable<T> növləri üçün genişləndirmə metodlarıdır. Onlar System.Linq.Enumerable və System.Linq.Queryable siniflərində müəyyən edilir. LINQ-da süzgəc, çeşidləmə, qruplaşdırma, toplama, birləşmə və s. kimi müxtəlif funksiyaları təmin edən 50-dən çox standart sorğu operatoru mövcuddur.

| **Klassifikasiya** | **Standard Sorğu Operatorları** |
| --- | --- |
| **Filtering** | Where, OfType |
| **Sorting** | OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse |
| **Grouping** | GroupBy, ToLookup |
| **Join** | GroupJoin, Join |
| **Projection** | Select, SelectMany |
| **Aggregation** | Aggregate, Average, Count, LongCount, Max, Min, Sum |
| **Quantifiers** | All, Any, Contains |
| **Elements** | ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault |
| **Set** | Distinct, Except, Intersect, Union |
| **Partitioning** | Skip, SkipWhile, Take, TakeWhile |
| **Concatenation** | Concat |
| **Equality** | SequenceEqual |
| **Generation** | DefaultEmpty, Empty, Range, Repeat |
| **Conversion** | AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList |