**C# nədir**

C# Microsoft tərəfindən .Net texnologiyası üçün inkşaf etdirilən modern bir tam obyekt yönümlü proqramlaşdırma(OOP) dilidir və C ailəsinə daxildir. C# datatype sensitivity’dir.

**Data type ranges (Verilənlərin tiplərinin aralıqları)**

byte (Yaddaşda tutduğu yer 1 byte) 0 to 256

sbyte (Yaddaşda tutduğu yer 1 byte) -128 to 127

short (Yaddaşda tutduğu yer 2 bytes) -32,768 to 32,767

ushort (Yaddaşda tutduğu yer 2 bytes) 0 to 65,535

int (Yaddaşda tutduğu yer 4 bytes) -2,147,483,648 to 2,147,483,647

long (Yaddaşda tutduğu yer 8 bytes) -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

ulong (Yaddaşda tutduğu yer 8 bytes) 0 to 18,446,744,073,709,551,615

float(Yaddaşda tutduğu yer 4 bytes) vergüldən sonra 6 və ya 7 rəqəm üçün nəzərdə tutulub

double(Yaddaşda tutduğu yer 8 bytes) vergüldən sonra 16 rəqəm üçün nəzərdə tutulub

boolean(Yaddaşda tutduğu yer 1 byte) true və ya false

char (Yaddaşda tutduğu yer 2 bytes) Simvollar və ya onlara uyğun olaraq ASCII kodları

Var keyword’ü ilə variable’a value mənimsədildiyi zaman compile olaraq datatype müəyyən edilir və biz həmin dəyişənə başqa bir yerdə fərqli datatype’dan bir value mənimsədə bilmərik. Bunu etsək compile time error verəcək.

Dynamic keyword’ü ilə variable’a hər dəfə müxtəlif datatype dəyər mənimsədilə bilər. Runtime olaraq datatype mənimsədildiyi üçün var keyword’dən fərqlənir.

Console.ReadLine(); ilə daxil edilən hər bir dəyər string type olaraq qayıdır.

Value type variable’ın default dəyəri 0’dır

Char type variable’ın default dəyəri 0’dır.

String type variable’ın default dəyəri null’dır.

**Arithmetic operators**(Hesab operatorları)

**/** - div bölmə adlanır , tam hissə götürülür

**%** - mod bölmə adlanır , qalıq hissə götürülür

**Increment** 🡪 ++

**Declerement** 🡪 --

int num = 10;

num++ && num-- 🡪 Postfix increment&decrement adlanır. Burada gedən proses özündən sonraki sətrə aid olur. Console.WriteLine(num++); yazarsaq ekrana **10** çıxacaq lakin artıq yaddaşda num **11** olaraq qalacaq və daha sonra num dəyişənini ekrana verərsək **11** kimi çıxacaq

++num && --num 🡪 Prefix increment&decrement adlanır. Burada həmin sətr daxilində artıq proses getmiş olur. Console.WriteLine(++num); yazarsaq ekrana **11** çıxacaq çünki artıq proses həmin sətrdə getmiş olacaq

**Assigment Operators**(Mənimsətmə operatorları)

a += b 🡪 a = a + b

a -= b 🡪 a = a - b

a \*= b 🡪 a = a \* b

a /= b 🡪 a = a / b

a %= b 🡪 a = a % b

**Comparison Operators**(Müqayisə operatorları)

Müqayisə operatorları bizə true və ya false dəyər qaytarır

== 🡪 bərabərdirmi?

!= 🡪bərabərliyin əksi, bərabər deyilmi yəni not equals

> 🡪 böyükdürmü

< 🡪 kiçikdirmi

>= 🡪böyük bərabərdirmi

<= 🡪kiçik bərabərdirmi

Bu operatorlar bizə boolean dəyər qayatırlar. Və əsasən if şərti daxilində və ya loop’larda istifadə olunurlar.

**Logical Operators**(Məntiqi operatorlar)

&& 🡪 and(və) deməkdir. Işləndiyi bütün şərtlər true olduğu halda true dəyər qaytarır

|| 🡪 or(və ya) deməkdir. Işləndiyi şərtlərin ən azı 1-I true olduğu halda true dəyər qaytarır

! 🡪 not(əks) deməkdir. Işləndiyi şərt true olarsa false , false olsarda true dəyər qaytarır.

**Condition**

**If Statements**

If(condition) {

Statement;

}

Statement2;

Bu ifadəyə if-then ifadəsi deyilir. Əgər verilən şərt doğru olarsa if daxilindəki ifadə yerinə yetirilir və daha sonra 2-ci ifadə yerinə yetirilir. Əks təqdirdə sadəcə 2-ci ifadə yerinə yetirilir.

If(condition) {

Statement;

} else{

Statement2;

}

Bu ifadəyə if-then-else ifadəsi deyilir. Əgər verilən şərt doğru olarsa sadəcə if daxilindəki ifadə yerinə yetirilir əks təqdirdə sadəcə 2-ci ifadə yerinə yetirilir.

if(condition) {

Statement;

} else if(condition2){

Statement2;

} else if(condition3){

Statement3;

} else{

StatementN;

}

Else-if ifadəsi müəyyən bir şərt ödənmədikdə verilə biləcək digər şərtə uyğun prosesin işə salınması üçün istifadə edilir

**Switch case**

Bizim təyin edtiyimiz dəyişənin müxtəlif qiymətlərə və ya dəyərlərə bərabər olma halı yoxlanılırsa if statement yerinə switch case’dən istifadə edilir. Əgər bizim dəyişənimizə heç bir case halı uyğun olmazsa default daxilindəki əməliyyatlar işə düşür.

switch(expression) {

case value1:

Statement;

break;

...

case valueN:

StatementN;

break;

default:

StatementDefault;

break;

}

**Switch case expression**

datatype variable = expression switch {

Value1ForExpression=> value,

...

ValueNForExpression=>value,

\_=>defaultValue

};

Switch case expression hər hansısa dəyişənə switch case’dən daha rahat və qısa yazılış ilə dəyər mənimsətmək üçün istifadə edilir. Default ‘\_’ simvolu ilə işarələnir.

Biz switch case expression vasitəsi ilə həmçinin müəyyən şərtləri aşağıdaki şəkildə yoxlaya bilərik.

Value1ForExpression => value əvəzinə, datatype variable2 when statement => value yazaraq əgər statement bizə true dəyərini qaytararsa variable’a value’nu mənimsədir.Burada variable2 bizim expression kimi göndərdiyimiz dəyərə bərabər olur və statement daxilində istifadə etmək üçündür.

>Value1ForExpression => value və ya <Value1ForExpression=> value bu yazılış isə o deməkdir ki expression olaraq göndərilən dəyər Value1ForExpression expression’dan böyükdürsə və ya kiçikdirsə value’a mənimsədilsin.

**Loop Statements**

Loop(dövr) qarşıya qoyulmuş məsələni müəyyən şərt daxilində müəyyən say qədər yerinə yetirmək üçün istifadə edilir.

**While**

while(condition) {

Statement;

}

Şərtimiz doğru olduğu müddətcə blok daxilindəki ifadə yerinə yetiriləcək.Bu da o deməkdir ki ilk olaraq hər dəfə gedib şərt yoxlanılacaq və true dəyər qayıdarsa prosess yerinə yetiriləcək.Bu səbəbdən While’a həm də ön şərtli dövr operatoru deyilir.

**Do While**

do{

Statement;

}while(condition)

Do while ilə while’ın fərqi odur ki, do while zamanı şərt yoxlanılmadan ilk başda proses 1 dəfə yerinə yetirilir daha sonra şərt yoxlanılır. Bu da o deməkdir ki verilən condition false olsa, while loop’da proses heç vaxt yerinə yetirilməyəcək amma do while’da isə proses 1 dəfə yerinə yetiriləcək. Bu səbəbdən Do While’a həm də son şərtli dövr operatoru deyilir.

**For**

for(dataType variable; condition; process1){

Process2;

}

For loop’u 3 hissəyə bölünür 1-ci hissə dəyişən təyin etmək üçün, 2-ci hissə şərt vermək üçün və sonuncu hissə müəyyən bir əməliyyatı yerinə yetirmək üçün istifadə olunur.Şərt ödəndiyi bütün hallarda For loop’un bloku daxilindəki proses işə düşür.

for(;;){

Process;

}

Bu yazılış infinite for loop adlanır. Müəyyən şərt daxilində loop break olunmazsa sonsuzadək davam edəcək.

**Foreach**

foreach(var item in collectionName) {}

Foreach loop’u vasitəsilə biz hər hansısa collection’un 1-1 elementlərinə müraciət edəbilərik. Bir sıra situasiyalarda foreach bizim köməyimizə çatır və for loop’undan daha sürətlidir.C# 8’dən sonra gəlib.

**Array**

DataType[] arrName = new DataType[Length]; və ya {value&variable1,...value&variableN};

Index’lənmə 0’dan başlayır və sonuncu index’də duran elementin indexi arrName.Length-1 olur.Array’lər resizeable deyildir bu səbəbdən biz ölçünü yalnız array’i yaradan zaman təyin edəbilərik və bir də dəyişə bilmərik. Array daxilindəki elementləri

**Keywords**

Break keyword’ü müəyyən bir şərt daxilində dövrü sonlandırmaq üçün və ya switch’dən çıxmaq üçün istifadə olunan bir ifadədir.

Continue keyword’ü vasitəsi ilə biz müəyyən şərt daxilində işləndiyi sətrdən sonra ki əməliyyatlar yerinə yetirilmədən loop’un əvvəlinə qayıtmaq üçün istifadə edilir.

Continue keyword’dən fərqli olaraq break keyword’ü loop’la yanaşı switch daxilində də işlənilir

Return keyword’ü işləndiyi yerdə daxilində yazıldığı methodu dəyandırır və özündən sonra gələn code sətrlərini işə salmır. Əgər method geriyə bir dəyər qaytarırsa return’ün qarışısında o type’dan dəyər yazılmalıdır.

Is null keyword’ü vasitəsi ilə biz ==null şərtini yoxlayıb geriyə true(əgər null olarsa) və ya false( əgər null’dan fərqli olarsa) dəyəri alırıq.C# 7’dən sonra gəldi.

Is not null keyword’ü vasitəsi ilə isə is null keyword’ün əksini yerinə yetirmiş olarıq. C# 9’dan sonra gəldi

??=value; keyword’ü vasitəsi ilə biz ==null şərtini yoxlayıb əgər geriyə true(əgər null olarsa) dəyəri gələrsə şərt yoxlanılan variable’a uyğun dəyər mənimsətmək üçün istifadə edirik.

Datatype variable = variable2?? value; əgər variable2 null deyilsə value’nu variable’a mənimsətmək üçün bu cür yazılış istifadə edilir.

**Methods / Functions**

Funksiya sərbəst yazıla bilən, dəyərlər göndərilib call oluna bilən bir code block’dur.Məqsədi isə Don’t Repeat Yourself(DRY principe).Biz hərhansısa bir code block’nu 2 və ya daha çox istifadə etsək o zaman həmin code block’nu funciton(method) halına salmaq lazımdır.

Hər hansısa class daxilində yəni hansısa type’a aid yazılan funksiya method adlanır.

Method və funksiya yerləşdiyi yerə və aid olmasına(Type’a) görə fərqlənir.Lakin hər 2’si də eyni işi yerinə yetirir.

**Method’un yazılışı**

accessModifier static&non-static returnType MethodName(parameters) {}

Method’un qəbul etdiyi dəyər parameter, method’a göndərilən dəyər isə argument adlanır.

ReturnType’ı olan method bütün statement’lərdə geriyə dəyər qaytarmalıdır.Əgər returntype’ı yoxdursa geriyə dəyər qaytarmır və yerinə void yazılır.Bu method void method adlanır.

**Method Optional Parameters**

Method’un qəbul elədiyi parametrə default dəyər vermək o deməkdir ki , default dəyəri parametrə set eləsin. Method call olunan zaman əgər default dəyər verilən parametrə hər hansısa bir arqument(dəyər) göndərilməsə default olaraq verilən dəyəri saxlasın və method daxilində istifadə eləsin.Əgər arqument göndərilərsə göndərilən arqumenti set eləsin və method daxilində arqumenti istifadə eləsin,yəni default dəyəri override eləsin.

Məsələn:

Public static void TestMethod(int param1 , int param2 = 5) {}

Biz TestMethod’u call edən zaman 2 arqument də göndərə bilərik 1 arqument də. Əgər 1 arqument göndərsək param2’yə set elədiyimiz default dəyər method daxilində istifadə ediləcək, 2 arqument göndərsək default dəyər override olunacaq və göndərilən arqument istifadə ediləcək.

Methodun istənilən sayda eyni typedan dəyər qəbul etməsi üçün params keyword’dən(optional parameter) istifadə edilir.Göndəriləcək olan dəyərlərin type’na uyğun olaraq həmin type’dan array parametrinin qarşısına params keyword’ü yazırıq.

Məsələn:

Public static void TestMethod2(params dataType[] arr) {}

Biz methodu çağıran zaman istənilən qədər verilən dataType’dan dəyər göndərə bilərik və sonra göndərilən dəyərlər parametr olaraq qəbul olunan array’in daxilinə yığılacaq.Və daxildə array’in elementləri kimi istifadə olunacaq

Optional parametr olaraq verilən parametrlər tələb olunan parametrlərdən sonra yazılmalıdır. Yəni optional parametr’in istifadə ardıcıllığı sonuncudan başlayaraq sağdan sola doğrudur.

**Method Signature**

Method signature, method’un adı method’un qəbul etdiyi parametrlərin sayı və method’un qəbul etdiyi parametrlərin type’na deyilir. Eyni class daxilində method signature’ləri eyni olan 2’ci bir method yaratmaq olmaz(compile error verir).

**Method Overloading**

Method overloading o deməkdir ki Class daxilində eyni adda olub, Method signature gözlənilərək yaradılan methodlar overload method adlanır.Yəni methodların adı eyni, qəbul etdiyi parametrlər‘in sayı və ya type‘ı fərqli olarsa bu overloading adlanır.

Method overloading static polymorphism adlanır. Və Compile zamanı baş verir.

Overload method‘larda qəbul edilən parametrlərə optional parametr versək method call olunan zaman ən uyğun variant işə düşəcək

Public static void printStr(string str) =>Console.WriteLine(str);

Public static void printStr(string str, string str2=”default value”) =>Console.WriteLine(str);

Public static void printStr(params string[] str) =>Console.WriteLine(str);

printStr(“testStr”); method‘u call olunan zaman görünən odur ki hər 3 overload halına da uyğundur, lakin ən uyğun variant sadəcə 1 parametr qəbul edən variantdır, optional parametr olan variantlar yox.

**Parametr və arqument**

Method’un qəbul elədiyi dəyər parametr, method call olunan zaman ona göndərilən dəyər isə arqument adlanır.

**Reference & Value type**

Reference type’ların ünvanı(referansları) stack yaddaşda özləri isə heap yaddaşda saxlanılır.Stack yaddaş heap yaddaşdan daha sürətlidir çünki burada datalar ardıcıl yaddaşda saxlanılır.Hər bir şey stack’dən oxunur.

Bütün number type’lar , char və bool type’lar,enumlar yəni struct reference olanların hamısı value type adlanır.

Objectlər,String’lər, array’lər delegate’lər interface’lər yəni Class reference olanların hamısı reference type adlanır.

Reference type dəyişənlərə hər hansısa bir method daxilində dəyişiklik olunarsa method işə düşdükdən sonra heap yaddaşda həmin dəyişəndəki dəyişiklik saxlanılacaq

**Ref & Out**

Method daxilində value type’da edilən dəyişiklik ümumi olaraq da(stack yaddaşda) dəyişilməsi üçün ref və out keyword’dan istifadə edilir.Hər zaman value type üçün istifadə edilmiş həmçinin reference type’lar üçün də istifadə edilə bilər.

Method parametr olaraq qəbul etdiyi dəyişənin əvvəlinə ref və ya out yazılır və arqument olaraq göndəriləndə də ref və ya out keyword’ü ilə göndərilir.

**Ref & Out fərqi**

Out keyword’ü istifadə etdikdə variable’a əvvəldə dəyər mənimsədilməsə də olar amma method daxilində mütləq variable’a dəyər mənimsədilməlidir.Ref keyword’dən istifadə zamanı isə mütləq variable’a əvvəldə dəyər mənimsədilməlidir yəni unassign olan dəyər göndərmək olmaz.

**OOP**

**Class və object məntiqi**

Class ramda yer tutmur. Biz bir class yaratdıqda yeni type yaratmış oluruq və bu typların objecti yarana bilər. Class‘dan inistance alınaraq object yaranır və bu object Class‘ın adını özünə type olaraq götürür.Class daxilində static olmayan hər şey objectə aid olduğu üçün bir başa heap yaddaşda saxlanılır.

Anonym object dedikdə var keyword‘ü ilə yaranmış objectlər başa düşülür. Var keyword‘ü yazılmasa anonym object yarana bilməz.Biz Anonym object‘ləri class yaratma ehtiyyacımız olmadıqda, hər hansısa objectən yalnız bir dəfə istifadə etməli olduğumuz hallarda yaradırıq

var objectName = new

{

Field1 = value,

...

FieldN = value

};

Set və Get anlayışı o deməkdir ki, hər hansısa bir object‘in hər hansısa field‘a & property‘ə value assign etmək set etmək deməkdir, object‘in hər hansısa field‘ın & property‘in dəyərini götürmək isə get etmək deməkdir.

This keyword‘ü vasitəsi ilə biz olduğumuz class daxilində onun objectini çağırmış oluruq.Biz class daxilində hər hansısa static olmayan property‘ə & field‘a müraciət etsək arxa tərəfdə this.property & this.field kimi işləyir.

**Constructor**

Constructor yalnız object inistance alan zaman işə düşən, return type‘ı olmayan method‘dur. Biz Class daxilində yazsaq da, yazmasaq da boş Constructor(parameterless) yəni heç bir dəyər qəbul etməyən Constructor compile zamanı əlavə edilir.Biz Constructor‘u method signature’ə riayət edərək overload edə bilərik.Constructor‘un adı daxilində yazıldığı Class‘ın adı ilə eyni olmalıdır.

Hər hansısa bir Constructor işə düşən zaman Class daxilindəki hər hansısa başqa bir Constructor‘u da işə salmaq istəsək :this(overload) yazaraq işə sala bilərik. Adətən constructor‘un constructor‘u çağırması field‘lara default dəyərlər set olunması üçün istifadə edilir.

Hər hansısa bir Constructor işə düşən zaman onun miras aldığı Class‘ın daxilindəki constructor‘u işə salmaq istəsək :base(overload) yazaraq işə sala bilərik.

**Object Oriented Programing(OOP) ‘in 3 əsas prinsipi vardır.**

* Inheritence
* Polymorphism
* Encapsulation

**Inheritence**

Inheritence anlayışı bir class‘ın digər class‘dan “miras” alması deməkdir.Burada miras anlayışı dedikdə, sub class‘ın super class‘daki method‘lara,property‘lərə & field‘lara yəni Class members‘ə müraciət edə bilməsinə, onları istifadə edə bilməsinə icazə verilməsi deməkdir.

Əgər miras alınan Class‘ın boş constructor‘u yoxdursa onun overload halı varsa subClass’dan inistance alınan zaman yəni öz constructor‘u işə düşən zaman :base(overload) constructor da işə salınmalıdır.

**Is a & has a**

“Is a” anlayışı o deməkdir ki, əgər “developer is a person” =true, o zaman developer class‘ı person class‘dan miras ala bilər.

“Has a” anlayışı o deməkdir ki, əgər “developer has a -pc” =true, o zaman pc object‘i developer class‘ın property & field‘ı ola bilər.

String Builder

**Delegate**

Method’un(function) parametr kimi hər hansısa bir method’a göndərilməsi callback function adlandırılır.

Biz hər hansısa function’ı parametr kimi bir method’a göndərmək istəyiriksə delegate’dən istifadə edirik.Delegate type’ın body’si olmur.

Delegate function’lara type vermək üçün istifadə edilir, hər hansısa bir funcsiton’un geriyə qaytardığı dəyərin type’ı, qəbul etdiyi parametrlərin sayı və type’ı delegate ilə eyni olarsa o zaman function delegate olaraq yaratdığımız type’da olacaq.

Delegate type bu şəkildə yazılır:

accessModifier delegate static&non-static returnType delegateTypeName(parameters);

yuxarıda deyilən şərtləri ödəyən bütün function’lar artıq delegateTypeName type’dan bir function olmuş olur və hər hansısa bir method’a parametr olaraq göndərmək istəsək type olaraq delegateTypeName func yazıb göndərə bilərik

Biz yaratdığımız delegate vasitəsi ilə hər hansı bir function’u bu şəkildə yarada bilərik.

delegateTypeName funcName = delegate(parameters) {

return variable;

};

biz anonym funksiyaları da delegate type olaraq yaza bilərik və qəbul olunan parametr sayəsində göndərilən arqumentin type’nı yazmaq məcburiyyətində qalmırıq.

Məsələn:

Public void method(delegateTypeName func)

Delegate bool TestDelegate(int num);

bool TestMethod1(int num) {return num>0} ilə

bool TestMethod2(int num) => num>0 eyni mənanı kəsb edir.

Biz hər hansısa methoda TestMethod1 və ya TestMethod2’ni TestDelegate type’dan bir function kimi göndərmək yerinə gəlib həmin method’a num => num>0 yazsaq əgər burada TestDelegate type’a görə num’ın int type olduğu bilinir və qarşısına dataType yazılmır.

Method(TestMethod1);

Method(TestMethod2);

Method(num => num>0);

Public delegate R DelegateTypeName<in T, out R>(T variable);

Bu yazılış o deməkdir ki bu method T type qəbul edəcək və R type returnType qəbul edəcək. T type parameter qəbul etsin və R type parameter return etsin.

Action

Biz action vasitəsi ilə delegate type’la yazdığımız methodun eynisi yazırıq lakin fərqləri odur ki, action hazır delegate’ləri istifadə etmək üçündür. Action’un 17 fərqli overload’ı mövcuddur

Delegate int TestDelegate<T>(T str);

TestDelegate<string> Method1 = delegate(string str) {

Console.WriteLine(str);

};

TestDelegate<string> Method2 = str=>Console.WriteLine(str);

Action<string> Method3 = str=>Console.WriteLine(str);

Function

Functionlarda əvvəldə olan qəbul etdiyi type’dır sonda olan isə return type’dır

Function<T , R>

Predicate

Boolean dəyər qaytarır

**Reflection**

Assembly class’ı namespace olaraq reflection’un daxilində yerləsir.

Assemly asmb = Assembly.GetExecutingAssembly();

Bizə hal hazırda olduğumuz proyektin assembly’sini verir yəni .exe file’ı verir.Burada asmb daxilində proyektin adı, version’u və s məlumatlar yerləşir.

Type type = typeof(ClassFullName) bizə type qaytarır.

Type type = obj.GetType() bizə type qaytarır.

Typeof GetType’dan daha üstündür çünki typeof vasitəsi ilə biz birbaşa type adını yazdığımız üçün bunu istifadə edərək gedir type’ı götürür bu səbəbdən static class’ların da type’nı götürə bilirik.

Type type = asmb.GetType(“Class Full Name”);

Bizə type tipindən class’ı(type) qaytarır və daxilində namespace ilə bilikdə adını göndərdiyimiz class’ı saxlayır

Type[] types = asmb.GetTypes();

Bizə Type tipindən bir array qaytarır və daxilində bizim proyektimizdəki class’ları saxlayır.Foreach’ə salıb bir-bir bütün elementlərə müraciət edə bilərik.

foreach(Type type in types) {

Console.WriteLine(type.Name);

}

type.Name bizə class’ların yəni type’ların adını qaytaracaq.

type.FullName bizə namespace ilə birlikdə adını qaytaracaq yəni root’nu.

Type.GetProperty(“Property Name”);

Bizə PropertyInfo tipindən property qaytarır və daxilində həmin property’in adını və dataType’ı saxlayır.

type.GetProperties()

Bizə PropertyInfo tipindən array qaytarır və daxilində həmin class(type) daxilindəki property’lərin adını dataType’ı saxlayır.

PropertyInfo[] properties = type.GetProperties();

foreach(PropertyInfo property in properties) {

Console.WriteLine(property.Name);

}

type.GetMethod(“methodName”)

Bizə həmin class’ın(type) method’nu qaytarır.

type.GetMethods()

Bizə həmin class’ın(type) method’larını qaytarır.

type.Field(“fieldName”)

Bizə həmin class’ın(type) field’nı qaytarır.

type.Fields()

Bizə həmin class’ın(type) field’larını qaytarır.

BURA MƏNƏ XATIRLATICI KİMİ İSTİFADƏ EDİLİR QAYDADAN XARİCDİR.

Paradigma

Array indexləri ardıcıl yaddaş blokunda saxlayır, bu səbəbdən indexə görə axtarış,çağırma o(1) sürətidir və ən sürətli axtarış array’ə aiddir.

Arrayi valueyə görə axtarış,çağırma sürəti o(n)dir(worst case)

\*Time complexity Big O notation

Anonym function

İnt type‘ında olan hər şey stack yaddaşda saxlanılmır

Proqramlaşdırma dilinin oxunma ardıcıllığı

1.C# intro , 2.C# intro-2 , 2.control structure, 3-Method.Parametr-Argument, 4.Method Overloading. Ref-Out 5.class, 6-Inheritence,Constructor,7-StringBuilder 20:00

20.Delegate 30:00

10.abstract 40:00

22.Collections 26:47

23.Reflection 40:00

REMOVE INSTERT

Using directive

Exception global olaraq proqramı dəyandırmır amma error dəyandırır.

Static classın memberləri də static olmalıdır.

Console.WriteLine($”stringIfadə {variable}”)

Console.WriteLine(”stringIfadə {0} ”,variable) – bu yazılış o deməkdir ki biz vergül vasitəsi ilə variable’larımızı göstəririk və index məntiqi ilə {index 0’dan başlayır} onlara müraciət edirik

“” {} ’

**Flow chart**

