7-Дәріс. C# тілінде класс әдістерін қолдану

Дәрістің мақсаты:

- C#-та әдістерді (құру, жариялау, шақыру) сенімді қолдану: қолжетімділік модификаторлары (public/private), қайтарым типтері, параметрлер, static.
- Параметр беру үлгілерін түсіну, әдіс денесі мен айнымалылардың әсер ету аймағын ажырату.
- return операторын екі режимде қолдану: мәнсіз (void) және мән қайтаратын әдістерде.
- Рекурсия ұғымын бекіту: базалық жағдай, рекурсивті шақыру тереңдігі, типтік мысалдар (факториал, Фибоначчи, ЕҮОБ).
- Кодты құрылымдаудың жақсы тәжірибелерін меңгеру: шағын, қайта қолданылатын әдістер, атау беру, модульдеу.

Негізгі тақырыптар

1. **Әдіс деген не?** Класс құрамындағы логикалық бірлік; C#-та барлық функция әдіс ретінде тек класс ішінде өмір сүреді. Негізгі кіру нүктесі — static void Main(...).
2. **Жариялау синтаксисі:**қатынасу_түрі қайтарылатын_тип Аты(параметрлер) { /* денесі */ } Қатынас модификаторы көрсетілмесе — private. Қайтарымы жоқ — void.
3. **static мәні:** Экземплярсыз шақыру; консольдік демоларда негізгі әдістерді static беру ыңғайлы.
4. **Параметрлер:** Формальды параметрлер мен нақты аргументтер; параметр аймағы — әдіс денесі; параметр арқылы дерек жеткізу және формattалған шығару.
5. **Мән қайтару:** return; (процедура) және return expr; (функция). Бір әдісте бірнеше return болуы мүмкін, бірақ артық бұтақтарды азайту — оқылымдылық үшін маңызды.
6. **Қарапайым демолар:** параметрсіз әдіс (adis1), параметрлі әдіс (adis2(string)), символдық сызықша салатын Line(int).
7. **Қайта қолдану және композиция:** Күрделі есепті шағын әдістерге бөлу (мысалы, ulken(x,y) арқылы 4 санның максимумын табу).
8. **Итерациялық vs рекурсивтік тәсілдер:**
   Факториалдың итерациялық нұсқасы (long Fact(int)), шектеулер (асып кету/тип таңдау).
   Рекурсивтік нұсқалар: Фибоначчи (Fib(n)), факториал (FactRk(n)), ЕҮОБ (Евклид алгоритмі).
9. **Рекурсия қауіптері мен тәртібі:** Базалық жағдай міндетті; стек тереңдігі; үлкен n кезінде өнімділік (экспоненциалды шақырулар) және жады.

Әдістерді жариялау және қолдану

ОБП пайда болуымен функционалды модульдердің архитектуралық рөлі артта қалды. C# тілін қамтитын ОБП тілдері үшін архитектруалық модульдің рөлін класс атқарады. Программалық жабдықтама жүйесі модульдерден тұрады, оларды кластар атқарады, бірақ осы модульдердің әрқайсысы белгілі бір мәліметтерді абстракциялай отырып, мазмұнды толтыра алады.

Программаны құру процесінде негізгі функция белгілі бір мәселелерді шешетін қосалқы функцияларға бөлінді. Бұл ыдырау процесі олар өте қарапайым функцияларды алғанға дейін жалғасады, оларды жүзеге асыруды программалау тілінің негізгі құрылымдарымен сипаттауға болады.

C# тілінде процедуралар мен функциялар класпен байланысқан, олар кластың қажетті функционалдығын қамтамасыз етеді және класс әдістері болып табылады. Объектіге бағытталған программалаудағы класс мәліметтің қандайда бір түрі ретінде қарастырылатындықтан, класта басты рөлді оның мәліметтері атқарады – кластың объектілерінің қасиеттерін өрістер анықтайды. Класс әдістері мәліметтерге «қызмет етеді», оларды өңдейді. C# нұсқасында процедуралар мен функциялар белгілі бір кластың әдістері ретінде ғана болады, олар кластан тыс жерде болмайды.

Мәліметтер (айнымалылар) мен олармен әртүрлі амалдар орындайтын әдістер (функциялар) программалардың екі негізгі бөліктері болып табылады. Бұған дейін қарастырылып келген программаларда мәліметтер болды, бірақ әдістер болған жоқ. Тек

мәліметтерден тұратын программалар жұмыс істей береді, бірақ оларда әдістер де болуы тиіс. Күрделі әрі көлемді программалар жазу кезінде программаны функцияларға – шағын бөліктерге бөлу арқылы жұмысты жеңілдетуге және бастапқы код көрнекілігін арттыруға болады.

Әдіс (функция, метод) – белгілі бір операциялар тобын атқаратын программа блогы**.** Функцияға мәліметтер береміз де, одан белгілі бір нәтижелер аламыз. Функцияның ішкі әрекеті программаның басқа бөліктеріне қатыссыз орындалады.

Програмада кем дегенде, бір функция болады және программадағы ол негізгі функцияның аты Main() болуы тиіс. Программаның орындалуы осы функциядан басталады. Main функциясының орындалуы барысында басқа функциялар біртіндеп шақырылады, ал олар одан да басқа функцияларды шақыруы мүмкін.

Әдісті анықтаудың жалпы формасы:

қатынасу_түрі қайтарылатын_тип аты(параметрлер_тізімі)

{ // әдіс тұлғасы (денесі) }

мұндағы қатынасу_түрі – бұл қатынасу модификаторы (public не private) ол осы әдісті программаның қандай бөліктерінен шақыруға болатынын білдіреді, модификаторды көрсету міндетті емес. Егер ол жазылмаса, онда ол жабық (private) болып саналады да, оны тек осы әдіс жарияланған класс ішінде ғана пайдалануға болады. Біз әзірше әдістерді ашық (public) етіп жариялаймыз, яғни оларды бір класс ішіндегі программаның кез келген жерінде қолдана береміз.

Қайтарылатын_тип әдіс нәтижесінің типін анықтайды. Егер әдіс мән қайтармайтын болса (экранға мәлімет қана шығарса), онда қайтарылатын типті void деп көрсету керек. Модификаторлар мен параметрлерді жазу міндетті емес болып табылады.

Әдісті анықтаудағы аты сөзі әдіске берілетін нақты атауды көрсетеді. Жариялау кезінде аты ретінде мағынасына қарай қолдануға болатын кез келген сөзді таңдай аламыз.

Ең соңғы параметрлер_тізімі – бұл үтірмен бөліне отырып, *типі* мен *идентификаторы* қатар көрсетілген, екі сөзден тұратын аргументтер (параметрлер). Параметрлер – әдісті шақыру кезінде оған берілетін *аргументтер* мәндеріне сәйкес айнымалылар тізбегі. Егер әдіс параметрлері болмаса, онда әдістің атынан кейін бос жақша тұрады.

Main әдісі мысалын қарастырып көрейік:

```
static void Main(string[] args)
{  Console.WriteLine("Программалау басталды!");  }
```

мұндағы static түйінді сөзі модификатор болып табылады. Сонан кейін қайтарылатын мән типі жазылады.

Тип ретінде тұрған void түйінді сөзі әдістің ешқандай мән қайтармайтынын көрсетеді. Мұндай әдісті *процедура* деп те атайды. Келесі тұрған әдіс атауы – Main және жақша ішінде параметрлер – string[] args орналасқан. Одан кейінгі жүйелі жақшалар ортасында орналасқандар – әдістің ішкі операторлары (денесі). Онда әдіс орындайтын барлық әрекеттер операторлар түрінде жазылады.

Әдістің қандайда бір мәнді қайтаруы немесе қайтармауына байланысты әдістер мән қайтаратын әдіс (функция), мән қайтармайтын әдіс (процедура) болып бөлінеді.

Әдістерде параметрлерді пайдалану. Әдісті шақырған кезде оған бір немесе бірнеше мәндер беруге болады. Әдіске берілетін мән *аргумент* деп аталады. Ал аргументті қабылдайтын айнымалыны формальді параметр немесе жай параметр деп атайды.

Параметрлер әдістің атынан кейін жай жақша ішінде жарияланады. Параметрлерді жариялау синтаксисі қарапайым айнымалыларды жариялау сияқты орындалады.

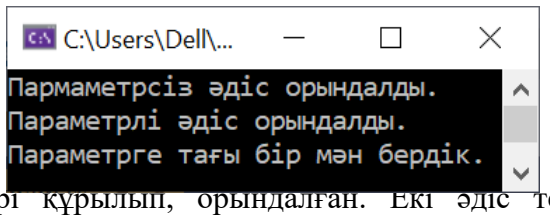
Параметрлердің әсер ету аймағы әдістің ішкі тұлғасы (денесі) болып табылады. Әдіске аргументтерді берудің ерекше жағдайларынан басқа сәттерде параметрлер кез келген басқа айнымалылар тәрізді жұмыс істей береді.

Келесі мысалда бір параметрсіз, бір параметрлі әдісті құрып, орындап көрейік:

```
using System;
```

```
using System.Text;
namespace Mysal
{
 class Program
  { static void adis1()                    // параметрсіз әдісті жариялау
    { Console.WriteLine("Пармаметрсіз әдіс орындалды."); }
   static void adis2(string t)   // параметрлі әдісті жариялау
    { Console.WriteLine(t); }

   static void Main(string[] args)
    {
     Console.OutputEncoding = Encoding.UTF8;
     adis1(); // 1-әдістің шақырылуы
     string t = "Параметрлі әдіс орындалды."; // t-ға мән беру
     string a = "Параметрге тағы бір мән бердік."; // a-ға мән беру
     adis2(t); // 2-әдістің t айнымалысы арқылы шақырылуы
     adis2(a); // 2-әдістің a айнымалысы арқылы шақырылуы
          Console.ReadKey();
    }
  }
}
```



Программаның орындалу нәтижесі:

Бұл мысалда adis1 және adis2 әдістері құрылып, орындалған. Екі әдіс те мән қайтармайды. Console.WriteLine әдісі арқылы мәтін экранға шығарылады. adis1 әдісінде параметр қолданылмайды. Adis2 әдісінде string типіндегі t параметрі беріліп, осы параметр сілтеме жасаған ақпаратты экранға шығарады. Main ішінде осы әдістер шақырылады. Онда adis2 әдісіне t және a айнымалыларын аргументтер ретінде береміз де, оны екі рет шақырып орындаймыз.

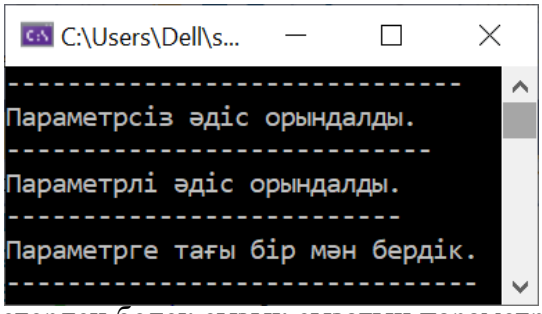
Енді осы мысалға аздаған өзгеріс енгізіп, adis1 және adis2 әдістері арқылы экранға шығатын мәтіндер арасына көлденең сызықтар орналастыратын Line әдісін қосайық:

```
using System;
using System.Text;
namespace Mysal
{
  class Program
  {
    static void adis1()                  // параметрсіз әдіс
    { Console.WriteLine("Параметрсіз әдіс орындалды."); }
    static void adis2(string t)  // параметрлі әдіс
    { Console.WriteLine(t); }
    static void Line(int m)      // параметрлі сызық сызу әдісі
    {
      for (int i = 1; i <= m; i++)
      { Console.Write('-'); }
      Console.WriteLine();
    }

    static void Main(string[] args)
    {
      Console.OutputEncoding = Encoding.UTF8;
      Line(30);   // 30 сызық сызу
      adis1();    // 1-әдістің шақырылуы
      Line(28);
      string t = "Параметрлі әдіс орындалды.";
      string a = "Параметрге тағы бір мән бердік.";
      adis2(t);   // 2-әдістің t айнымалысы арқылы шақырылуы
      Line(26);   // 28 сызықша сызу
      adis2(a);   // 2-әдістің t айнымалысы арқылы шақырылуы
```

```
            Line(31);   // 28 сызықша сызу
            Console.ReadKey();
        }
    }
}
```

Программаның орындалу нәтижесі:



Бұл мысалда алдыңғы программадағы әдістерден бөлек сызық сызатын параметрлі Line әдісі енгізілген. Line әдісінде m параметрі бар. Оның 1-ден m-ге дейін, for циклінің әр қадамы орындалған сайын, экранға '-' таңбасын шығарып отыратынын көруге болады. Соның нәтижесінде әдісті шақырған кезде параметр аргументі ретінде қай санды енгізсек, сонша сызықшалар сызылады. Мұнда line әдісі 4 рет орындалып, adis1 және adis2 әдістерінің берілген аргументті қабылдауы арқылы жолдар арасына көлденең сызықшалар орналастырған.

Әдістен кері қайту және рекурсияны пайдалану

Әдістен кері қайту екі түрлі жолмен жүзеге асады.

1) Әдістің ішкі операторларын (тұлғасын) жабатын жүйелі жақша кездескенде, мысалы, ол жоғарыдағы программада сызық сызатын Line() әдісінде орын алып тұрған болатын.

2) Әдіс ішінде return операторы орындалғанда жүзеге асады. Бұл return операторының да екі түрі болады, олар:

а) void типіндегі мән қайтармайтын әдістерде,

ә) нақты мәндер қайтаратын әдістерде кездеседі.

Жалпы әдісті аяқтау үшін return операторының келесі формаларының бірі пайдаланылады:

```
return;            // соңында ешқандай мән жоқ
return 1;                  // соңында сан бар
return (x+y);   // соңында өрнек бар
```

return операторы орындалғанда, жұмысты басқару программаның әдісті шақырған бөлігіне қайтарылады да, әдіс ішіндегі қалған операторлар орындалмайды. Мысал ретінде келесі әдісті қарастырайық.

```
public void adis()
{
        int i;
        for(i=0;  i<10;  i++)
        {
            if (i == 5)  return;  // 5-қадамда циклді аяқтау
            Console.WriteLine(i);
        }
}
```

Бұл мысалда for циклінің 5 толық қадамы орындалады да, i айнымалысының мәні 5-ке тең болғанда, әдістен кері қайту жүзеге асады.

Бір әдісте, одан шығудың бірнеше нұсқалары қажет болған сәттерде, бірнеше return операторлары болуы мүмкін. Мысалы:

```
public void adis()
{
  // ...
  if (done)  return;
  // ...
  if (error) return;
}
```

Бұл мысалда әдістен кері қайту екі жағдайда: әдіс жұмысын аяқтағанда немесе қате пайда болғанда, жүзеге асады. Бірақ әдістен кері қайту сәттері көбейген сайын кодтың құрылымы бұзылуы мүмкіндігі де артады.

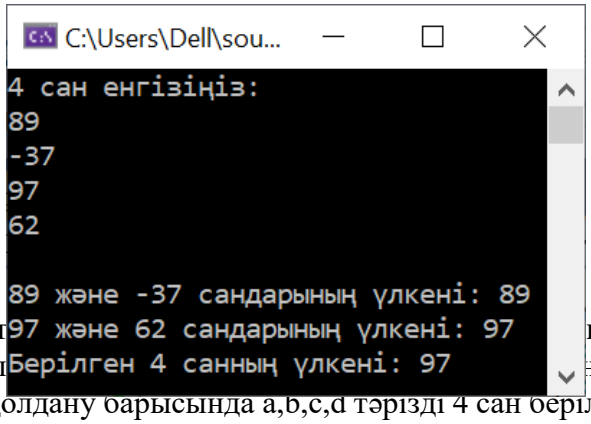
Сонымен, тағы да еске сала кетейік: әдістен кері қайту екі мүмкіндік арқылы: әдістегі ең соңғы жүйелі жақшаға жеткен кезде немесе return операторы орындалған сәтте жүзеге асады.

Return түйінді сөзінен кейін сан, өрнек, айнымалы тұратын әдістерді алдағы мысалдарда көрсететін боламыз. Келесі мысалда екі санның үлкенін табатын әдіс арқылы берілген 4 санның ең үлкенін табайық:

```
// 4 санның үлкенін табу
using System;
using System.Text;
namespace Mysal
{
  class Program
  {
```

```
    public static int ulken(int x, int y)
    {
      if (x > y) return x;    //  әдісті 1-аяқтау  тәсілі
      else return y;          //  әдісті 2-аяқтау  тәсілі
    }
    static void Main(string[] args)
    {
      Console.OutputEncoding = Encoding.UTF8;
      int a, b, c, d, max_a_b, max_c_d, max;
      Console.WriteLine("4 сан енгізіңіз: ");
      a = int.Parse(Console.ReadLine());
      b = int.Parse(Console.ReadLine());
      c = int.Parse(Console.ReadLine());
      d = int.Parse(Console.ReadLine());
      max_a_b = ulken(a, b);   // a және b сандарының үлкенін табу
      max_c_d = ulken(c, d);   // c және d сандарының үлкенін табу
      max = ulken(max_a_b, max_c_d); // 4 санның үлкенін табу
      Console.WriteLine();
      Console.WriteLine(a+" және "+b+" сандарының үлкені: " + max_a_b);
      Console.WriteLine(c+" және "+d+" сандарының үлкені: " + max_c_d);
      Console.WriteLine("Берілген 4 санның үлкені: " + max);
      Console.ReadKey();
    }
  }
}
```

Программа жұмысының нәтижесі:



Бұл мысалда екі санның үлкен табат параметр ретінде берілген екі сан салыстырылып, сол нде return операторы арқылы кері қайтарады. Әдісті қолдану барысында a,b,c,d тәрізді 4 сан беріліп, ulken әдісін шақыру арқылы алдымен a және b, сонан соң c және d сандарының үлкенін жеке-жеке тауып аламыз да, ulken әдісін үшінші рет шақырғанда, алдыңғы екі әдістің нәтижелерін параметрлер ретінде бере отырып, 4 санның ең үлкен мәнін табамыз.

Енді берілген санның факториалын есептейтін әдіс құрайық. Берілген n бүтін санының факториалы 1-ден n-ге дейінгі оң бүтін сандар көбейтіндісі, яғни n!=1*2*3*4*…*(n-1)*n болады. Мысалы, үштің факториалы 3! = 1*2*3 = 6.

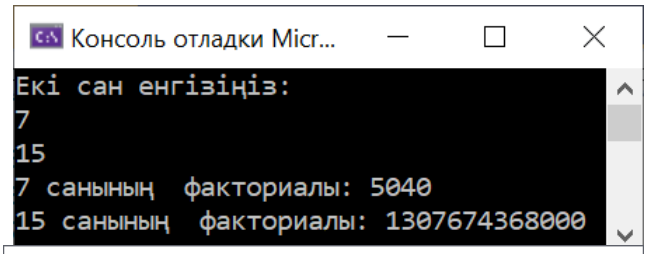
Теріс сандардың факториалы болмайды, бірақ 0!=1 болып есептеледі.

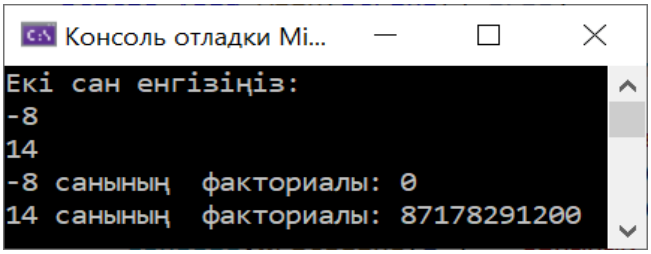
Келесі мысалда екі сан енгізіп, солардың факториалын есептейтін әдіс құрамыз:

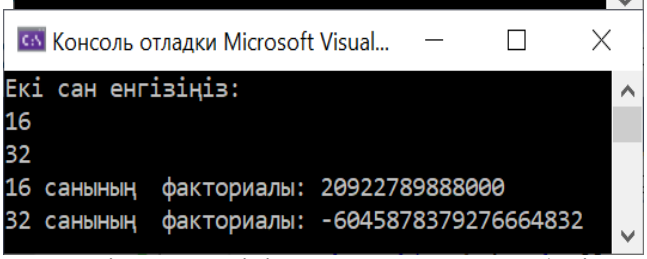
```
using System;
using System.Text;
namespace Mysal
{
  class Program
  {
    static long Fact(int k)              // Fact әдісін құру
    {
      if (k < 0) return 0;
      if (k == 0) return 1;
      long p = 1;
      for (int j = 1; j <= k; j++) p *= j;
      return (p);
    }
    static void Main(string[] args)
    {
```

```
            Console.OutputEncoding = Encoding.UTF8;
            int a, b;
            Console.WriteLine("Екі сан енгізіңіз: ");
            a = int.Parse(Console.ReadLine());
            b = int.Parse(Console.ReadLine());
            Console.WriteLine(a + "санының  факториалы: " + Fact(a));
            Console.WriteLine(b + "санының  факториалы: " + Fact(b));
        }
    }
}
```

Программа жұмысының нәтижелері:







Бұл мысалда сандардың факториалын есептейтін Fact әдісі құрылған. Мұнда берілген бүтін сан 0-ден кіші болса, әдіс 0-ді, сан 0-ге тең болса, 1-ді қайтарады. Қалған жағдайларда 1-ден бастап k параметрінің мәніне дейінгі сандарды көбейтіп, p айнымалысына меншіктеп отырып, цикл аяқталғанда, p айнымалысының мәнін кері қайтарады. Әдісті шақыру барысында, a және b сандарының факториалдары есептеледі. Нәтижелердегі 32 санының теріс мән беретін себебі, ол мән бүтін сандардың long типіне симайтын өте үлкен сан шығатынын білдіреді, мұндай шамалар үшін нақты сандар (double) типін қолдану керек.

Рекурсияны пайдалану. Әдіс, қажеттілік болған жағдайда, өзін өзі шақырып та жұмыс істей алады. Әдістің өзін өзі шақыру процесі рекурсия деп аталады, ал ондай әдістер – рекурсивтік әдістер болып табылады.

Рекурсивтік әдісте берілген әдіс осы әдістің өзін шақыратын болады. Мұндай тәсіл программалау тілдерінің бәрінде де қолданылады.

Фибоначчи тізбегі. Рекурсивтік түрде шығарылатын мысалдардың бірі математикада кең қолданылатын Фибоначчи сандары қатарының есептелуі болып саналады. Ортағасырлық итальяндық математик Леонардо Фибоначчидің (шамамен 1170 жылы туған) есімімен аталған Фибоначчи сандары төменде келтірілген тізбекпен беріледі:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

Бұл тізбектің екінші санынан кейін әрбір келесі саны алдыңғы екі санның қосындысынан тұрады. Фибоначчи тізбегі мынадай түрде анықталады.

Егер $n = 0$, онда $\text{Fib}(n) = 0$.
Егер $n = 1$, онда $\text{Fib}(n) = 1$.
Егер $n > 1$, онда $\text{Fib}(n) = \text{Fib}(n - 1) + \text{Fib}(n - 2)$.

Фибоначчи тізбегіндегі n-санды есептеудің рекурсивтік функциясы төменде келтірілген.

```
static int Fib(int n)
 { if (n == 0) return 0;
   else if (n == 1) return 1;
       else return Fib(n - 1) + Fib(n - 2);
 }
```

Бұл функцияның екі базалық жағдайы: $n = 0$ және $n = 1$ бар екендігін ескеріңіз. Олардың кез келгенінде функция рекурсивтік шақыруды жасамай, мән қайтарады. Келесі программа осы функция арқылы Фибоначчи тізбегіндегі алғашқы m санды көрсетеді.

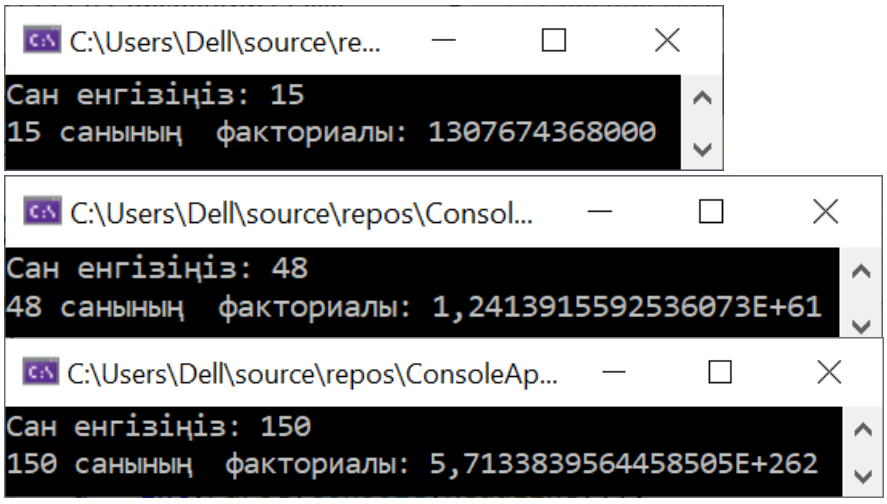
```
using System;
using System.Text;
namespace Mysal
{ // рекурсия Фиббоначи сандары
  class Program
```

```
{ // Фиббоначи сандары
  static int Fib(int n)
  {
    if (n == 0) return 0;
    else if (n == 1) return 1;
        else return Fib(n - 1) + Fib(n - 2);
  }
  static void Main(string[] args)
  {
    Console.OutputEncoding = Encoding.UTF8;
     int m;
     Console.Write("Фибоначчи қатарының неше санын шығарасыз: ");
     m = int.Parse(Console.ReadLine());
     Console.WriteLine("Фибоначчи қатарының алғашқы "+m+" саны:");
     for (int k = 0; k < m; k++)
        Console.Write(Fib(k) + "  ");
     Console.ReadKey();
    }
  }
}
```

Рекурсияның тағы бір жиі кездесетін классикалық мысалы факториалды есептеу болып саналады. Келесі мысалда енгізілген санның факториалын рекурсивті әдіс арқылы есептеу көрсетілген, онда сан факториалының нәтижесі double типінде берілген, ең соңғы нәтиже үлкен сандар факториалдары көрсетілуінің бұл типте де шектеулі екендігін білдіреді:

```
using System;
using System.Text;
namespace Mysal
{
  class Program
  {
    static double FactRk(int n)
    {
      double result;
      if (n == 1) return 1;
      result = FactRk(n - 1) * n;
      return result;
    }
    static void Main(string[] args)
    {
      Console.OutputEncoding = Encoding.UTF8;
      Console.WriteLine("Сан енгізіңіз: ");
      int a = int.Parse(Console.ReadLine());
      Console.WriteLine(a + " санының  факториалы: " + FactRk(a));
      Console.ReadKey();
    }
  }
}
```

Программа жұмысының нәтижелері:



Мұндағы рекурсивтік FactRk() әдісінде, егер FactRk() әдісінің аргументі 1 болса, оның нәтижесі де 1 болады, басқа кездерде ол FactRk (n-1) *n көбейтіндісін береді. Бұл көбейтіндіні есептеу үшін FactRk() әдісі n-1 аргументі арқылы шақырылады. Осы процесс n аргументі 1-ге тең болғанша орындалады, соңында алдыңғы есептеулер мәні алынып, барлығы көбейтіледі.

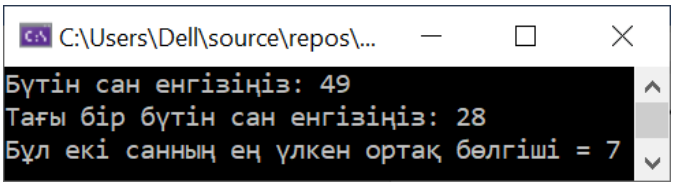
Ең үлкен ортақ бөлгішті табу. Рекурсияның келесі мысалы екі санның ең үлкен ортақ бөлгішін (ЕҮОБ немесе GCD - greatest common divisor) есептеу болып табылады. Екі оң бүтін *x* және *y* сандарының ЕҮОБ-і былайша анықталады.

Егер *x* -ті *y* -ке қалдықсыз бөлуге болса, онда gcd (*x*, *y*) = *y*.
Олай болмаған жағдайда gcd (*x*, *y*) = gcd (*y*, *x*/*y* бөлгендегі қалдық).

Бұл анықтама *x* және *y* сандарының ЕҮОБ-і, егер *x*/*y* бөлу қалдығы болмаса, *y* санына тең екенін білдіреді. Бұл шарт базалық жағдай болып табылады. Олай болмаған жағдайда, жауабы: *y* саны мен *x*/*y* саны қалдығының ЕҮОБ-і болып табылады. Келесі программада ЕҮОБ есептеудің рекурсивтік әдісі келтірілген.

```
using System;
using System.Text;
namespace Mysal
{ // Рекурсия: ЕҮОБ - ең үлкен ортақ бөлгіш табу
  class Program
  { static int gcd(int x, int y)
    { // Екі санның ең үлкен ортақ бөлгішін
      // (ЕҮОБ немесе GCD) табатын рекурсия
      if (x % y == 0) return y;
      else return gcd(x, x % y);
    }
    static void Main(string[] args)
    {  Console.OutputEncoding = Encoding.UTF8;
       int num1, num2;
       Console.Write("Бүтін сан енгізіңіз: ");
       num1 = int.Parse(Console.ReadLine());
       Console.Write("Тағы бір бүтін сан енгізіңіз: ");
       num2 = int.Parse(Console.ReadLine());
       Console.Write("Бұл екі санның ең үлкен ортақ бөлгіші = ");
       Console.WriteLine(gcd(num1, num2));
       Console.ReadKey();
    }
  }
}
```

Программа жұмысының нәтижесі:



Программалардағы рекурсивтік әдістер солардың итерациялық эквиваленттеріне қарағанда, ұзағырақ орындалады, өйткені әдістер жиі шақырылған кезде жүйелік ресурстар көбірек пайдаланылады.

Егер осындай шақырулар көбірек болып кетсе, жүйелік стек (жады) толып кетеді де, аластама пайда болуы мүмкін. Бірақ бұл тек рекурсивтік процедура дұрыс орындалмаған кезде ғана орын алады.

Бақылау сұрақтары

1. C#-та «функция» неге міндетті түрде класс ішіндегі әдіс ретінде анықталады?
2. public, private айырмашылығы қандай? Қайсысы әдепкі (default)?

3. void және мән қайтаратын әдістердің айырмасы неде? Қай кезде қандайын таңдаған дұрыс?
4. static әдіс пен экземпляр әдісін салыстырыңыз. Main неге static?
5. Параметр деген не? Формальды параметр мен нақты аргументті ажыратыңыз.
6. return операторын қолданудың екі тәсілін мысалмен көрсетіңіз.
7. Бір әдісте бірнеше return қолданудың артықшылығы/қаупі қандай?
8. Line(int m) секілді «утилиталық» әдістер не үшін керек? Оларды қалай қайта қолданамыз?
9. ulken(int x,int y) көмегімен 4 санның максимумын табудың логикасын түсіндіріңіз.
10. Итерациялық факториал әдісінде қандай тип қолдану орынды? Неліктен int аздық етеді?
11. Рекурсиядағы базалық жағдай деген не және не үшін керек?
12. Фибоначчи рекурсивтік әдісі неге баяу? Оны қалай жеделдетуге болады (қысқаша: есте сақтау, итерация)?
13. ЕҮОБ табудың рекурсивтік Евклид алгоритмінің идеясы қандай?
14. Рекурсияда стек толуы қандай жағдайда болады? Оны қалай алдын аламыз?
15. Әдістерді жобалауда атау беру, параметр саны, бір жауапкершілік принципі туралы 3 ереже атаңыз.

Ұсынылатын әдебиеттер

1. Колдаев, В.Д. Основы алгоритмизации и программирования: Уч. пособие/ под ред. проф. Л.Г.Гагариной. -М.: ИД «ФОРУМ»: ИНФРА-М,2015.-416 с.
2. Бөрібаев Б. Компьютердің арифметикалық негіздері: Жоғары оқу орындары студенттеріне арналған оқу құралы. -Алматы: Қазақ университеті, 2009. -80 б.
3. Бөрібаев Б. Алгоритмдеу, мәліметтер құрылымы және программалау тілдері: оқулық. – Алматы: Қазақ университеті, 2012. -235 б.
4. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. -М.: Мир, 2011.-360с.
5. Бөрібаев Б., Абдрахманова М. C# тілінде программалау (практикалық курс): оқу құралы –Алматы: Қазақ университеті, 2018. -258 б.
6. Troy Dimes. C# Programming for Beginners. Paperback – January 25, 2015.
7. Шилдт Г. C# 4.0: полное руководство. М.: ООО ”И.Д.Вильямс”, 2017. -1056 с.
8. Джуст В. Разработка обслуживаемых программ на языке C#. СПб.: Изд. дом «Питер», 2017.
9. Шарп Д. Microsoft Visual C#. Подроб. руководство. 8-е изд.С-Петербург: Изд. дом «Питер», 2016.
10. A. Troelsen, P. Japikse. Pro C# 7: With .NET and .NET Core. – Apress, 2017. -1372 p.