Қазақстан Республикасы Ғылым және жоғары білім министрлігі

Д. СЕРІКБАЕВ АТЫНДАҒЫ ШЫҒЫС ҚАЗАҚСТАН ТЕХНИКАЛЫҚ УНИВЕРСИТЕТІ

ЖҮЙЕЛІК БАҒДАРЛАМАЛАУ

Жомартқызы Г., Сулейменова Л.Р.

6В06102 - «Есептеуіш техника және бағдарламалық қамтамасыз ету» бакалавриат мамандығына арналған дәрістер кешені

Өскемен
Усть-Каменогорск
2024

МАЗМҰНЫ

1 C# ТІЛІНДЕ DLL ЖИНАҚТАРДЫ ҚҰРУ

1.1 Жүйелік бағдарламалаудың негізгі ұғымдары

1.1.1 Операциялық жүйелер ұғымы және олардың міндеті

Компьютер ұғымына физикалық, аппарттық, логикалық және ақпараттық ресурстар кіреді.

Физикалық ресурстар ол компьютердің құрамына кіретін құрылғылар – процессор, енгізу-шығару құрылғысы, жады және т.б.

Логикалық ресурстарға омпьютер жұмыс істейтін деректер, бағдарламалар жатады.

Компьютердің барлық ресурстары жүйелік ресурстар деп аталады.

Осыған байланысты операциялық жүйені компьютердің жүйелік ресурстарына қатынау және оларды басқаруды қамтамасыз ететін бағдарламалар кешені деп анықтауға болады.

Компьютерде ОЖ басқаруымен жұмыс істейтін бағдарламалар қолданушылар қолданушылар бағдарламалары деп аталады.

Бір тапсырманы шешуге арналған қолданушылар бағдарламалары қосымша деп аталады.

Егер ОЖ тек бір қолданушылар бағдарламасын орындауға мүмкіндік берсе, онда ол ОЖ бір қолданушы немесе бір бағдарламалық деп аталады.

Егер ОЖ бірнеше қолданушылар бағдарламасын орындауға мүмкіндік берсе онда ол ОЖ көп қолданушы немесе мультибағдарламалық деп аталады.

Егер ОЖ тек бір процессорлы компьютерде жұмыс істей алса, онда ол бірпроцессорлы ОЖ деп аталады.

Егер ОЖ аппараттық ресурсында бір немесе бірнеше процессор бар бола алатын компьютерде жұмыс істей алса, онда ол мультпроцессорлы ОЖ деп аталады.

Қолданушының катынасынсыз нақты уақыт режимінде жұмыс істейтін ОЖ нақты уақыт ОЖ деп аталады. Ондай ОЖ кейбір технологиялық процесстерді басқаруға арналған.

1.1.2 Қолданбалы программалаудың Win32 API интерфейсінің ұғымы және міндеті

Әрбір ОЖ қолданушыға өзінің барлық жүйелік ресурстарын қолдануға мүмкіндік береді - ОЖ басқаруымен жұмыс істейтін қосымшаларды бағдарламалауға қажетті типтер, константалар, айнымалылар, функциялар және класстар жиындары кіретін қолданушы интерфейсі.

Осы кезде жұмыс істейтін барлық Windows ОЖ-дің қосымшаларды бағдарламалау интерфейстері бірдей - Win32 API (Application Programming Interface).

Win32 API интерфейсі тек қана қолданбалы бағдарламалауда да ғана емес, жүйелік бағдарламалауда да қолданылады.

Win32 API функцияларын санаттарға бөлуге болады:

- базалық қызметтер (Base Services);
- ортақ басқару элементтерінің кітапханасы (Common Control Library);
- графикалық құрылғылардың интерфейсі (Graphics Devices Interface);
- желілік қызметтер (Network Services);
- қолданушы интерфейсі (Uses Interface);
- Windows қатынауды басқару (Windows Access Control);
- Windows қабықшасы (Windows Shell);
- Windows жүйесі туралы ақпарат (Windows System Information).

Қолданбалы бағдарламаларды жазумен байланысты пәндерде, негізінен қолданушы интерфейсінің функциялары оқытылады.

Желілік қызметтің функциялары жергілікті желілердің жұмысымен байланысты пәндерде оқытылады.

Графикалық құрылғылар интерфейсі түрлі ойын бағдарламаларын жазған уақытта қолданылады.

Win32 API басқа функцияларының тағайындалулары мен қолданыстары негізінен «Жүйелік бағдарламалау» пәніде оқытылады. .

1.1.3 Windows объектілері мен дескрипторлары түсініктері

Win32 API функцияларын қолдану объетілерді қолдануға негізделген, әрі бұл уақытта объектіні класс айнымалы деген объектінің «классикалық» анықтамасы Win32 API функцияларын әзірлегеннен соң қолданыла бастағанын есте сақтау керек.

Windows-та объект деп жүйелік ресурсы бар мәліметтер құрылымы аталады. Іс жүзінде бұл объект Windows бөлінген және объект туралы ақпарат сақталған жады облысы. Объектілер арнайы функциялардың көмегімен құрылады және оларға қатынау тек ОЖ функцияларының көмегімен ғана мүмкін. Win32 API үш санаттың объектілерін құра алады:

– қолданушы интерфейсінің объектісі (User);

–графикалық құрылғылар интерфейсінің объектісі (Graphics Devise Interface);

– ОЖ ядросының объектілері (Kernel).

Біз тек ОЖ ядросының объетілерін ғана қарастыратын боламыз.

Объектіні құруда оған атау (идентификатор) беріледі, ол дескриптор (handle) деп аталады. ОЖ объект дескрипторы объектінің мекені және объект типін идентификациялау құралдары бар кестегі жазба болып табылады.

Win32 API дескриптордың типі HANDLE болады

Объектіге қатынаған уақытта оның десрипторын өрсету қажет.

Объектімен жұмыс істеп болған соң оны жабу керек.

1.1.4 Динамикалық қосылатын жинақтар түсінігі

«Динамикалық қосылатын жинақтар (dynamic-link libraries, DLL)– Windows ОЖ ең алғашқы нұсқасынан бастап оның іргетасы. DLL-да Win32 API функцияларының барлығы бар. Ең мағызды үш DLL: Kernel32.dll (жадыны, үдерістерді және ағындарды басқару), User32.dll (қолданушы интерфейсін қолдау, оның ішінде терезелерді құру және хабарламаларды жіберу функцияларымен байланыстылары да) және GDI32.dll (графика және мәтінді шығару).

Windows-та функциялары мамандандырылған тапсырмаларды орындауға арналған басқа да DLL бар. Мысалы, AdvAPI32.dll, мұнда реестрмен жұмыс және оқиғаларды тіркеу объектілерін қорғауға арналған функциялар бар, ал ComCtl32.dll стандартты басқару элементтерін қолдайды.

Динамикалық қосылатын жинақтар не үшін керек? Міне, DLL қолдану керектігінің кейбір себептері ғана:

– қосымшаның қымзет етуін кеңейту. DLL-ді үдерістің адрестік кеңістігіне динамикалық түрде жүктеуге болады, ал бұл қосымшаға одан қандай әркеттер қажет екенін анықтап, керекті кодты жүктеуге мүмкіндік береді. Сондықтан, бір компания қандай да бір қосымшаны құрып, оның қызмет етуін басқа компаниялардың DLL есебінен кеңейте алады;

– түрлі бағдарламалау тілдерін қолдану мүмкіндігі. Сізде қосымшаның қандай да бір бөлігін қай тілде жазу таңдауы бар;

– жобаны қарапайым басқару. Егер бағдарламалық өнімді әзірлеу барысында оның жекелеген модульдерін әртүрлі топтар құрса, онда DLL қолданған уақытта бұл сияқты жобаны басқару әлдеқайда жеңіл;

– жадыны үнемдеу. Егер бір DLL бірнеше қосымша пайдаланса, онда оперативті жадыда оның осы қосымшалардың барлығына қолжетімді болатын бір ғана данасы сақталынады.

– ресурстарды бөлу. DLL-да диалогтық терезенің, жолдың шаблондары, белгілер және биттік карталар (растрлық суреттер) сияқты ресурстар болуы мүмкін. Бұл ресурстар кез келген бағдарлама үшін қолжетімді болады;

– түрлі платформалардың ерекшеліктерімен байланысты мәселелерді шешу. Windows-тың әртүрлі нұсқаларында әртүрлі функциялар жиыны бар. Көбіне әзірлеушілерге олар өздері қолданып отырған нұсқасында бар жаңа функциялар керек. Егер бұл функциялар сіздің Windows нұсқаңызда жұмыс істемейтін болса, онда Сіз бұл сияқты қосымшаны іске қоса алмайсыз: жүктеуші оны іске қосудан бас тартады. Бірақ, егер бұл функциялар жеке библиотекада (DLL) болатын болса, онда Сіз бағдарламаны тіптен Windows ертеректегі нұсқасында да жүргізесіз (Джеффри Рихтер Мамандарға арналған Windows, 476-477 б.).

2 C# ТІЛІНДЕГІ НҰСҚАҒЫШТАРДЫ ПАЙДАЛАНУ

2.1 C# тіліндегі биттік операциялар

2.1.1 Санақ жүйелері

Ақпаратты көрсетудің негізгі екі тәсілі бар – үзіліссіз немесе аналогтық формада және дискреттік немесе цифрлік формалардың көмегімен ақпаратты көрсету.

Үзіліссіз форма тірі табиғатта ақпаратты бейнелеудің негізгі формасы болы табылады, мысалы, дыбыстық сигналдар, дәмдік сапалар, ауру сезімдері, түс реңдер және т.б., міне бұлардың барлығы да аналогтық сигналдар болып табылады. Көбіне, үзіліссіз сигналдар жазықтықта уақытқа тәуелді функцияның координаттарында үзіліссіз графиктер ретінде бейнеленеді.

Ақпаратты бейнелеудің дискреттік формасы көбінесе техникалық құрылғыларда қолданылады, мысалы, автоматика жүйелерінде, компьютерлерде. Уақытқа тәуелді фнкция графигінде дискреттік форма деңгейлердің «қабаттары» сияқты түрде бейненеледі, әрі бір деңгейден келесі деңгейге өту өте жылдам уақыт (лезде) ішінде орындалады, мысалы, қандай да бір орындалушы механизнің жұмысы екі деңгей – қосулы немесе өшулі деңгейлерімен көрсетіле алады.

Сигналдың аналогтық формасының ақпараттылығы аса жоғары (үзілісісіз функциялар өлшеніп отырған дәлдікке байланысты миллиардтаған түрлі мәндері қабылдай алады), бірақ кедергілерден қорғалу деңгейі өте төмен болады (сыртқы кедергілердің әсер етуі сигнал формасының аздап өзгерілуіне әкелуі мүмкін). Бұл әсерлердің тірі табиғат үшін еш мәні жоқ, бірақ техникалық құралдар үшін бұл сияқты кедергілер болмауы керек.

Сондықтан, техникалық құрылғыларда, мысалы компьютерлерде ақпаратты бейнелеудің тек дискреттік түрі қолданылады, олардың шінде ең көп қолданысқа ие болғаны екілік форма (кедергілерден қорғалу деңгейі жоғары болғандықтан).

Екілік формада ақпарат екі деңгейдің көмегімен беріледі, олар – нөл және бір (сигнал бар, сигнал жоқ). Сигналды бейнелеудің екілік формасының кедергілерден қорғалу деңгейі ең жоғары деп саналынады.

Компьютерлерде ақпарат бірлігі ретінде бит (бір разряд) қолданылады, онда 0 немесе 1 жазуға болады. Сегіз бит бір байтты құрайды. Байт – көптеген компьютерлердегі адрестелінетін минималды ақпарат бірлігі. Ақпараттың тағы бір өлшемі ол сөз. 32 разрядтық кмпьютерлер үшін сөзде 32 бит немесе 4 байт бар, ал 64 разрядтық кмпьютерлер үшін сөзде 64 бит немесе 8 байт бар,

Компьютердегі басқа да операциялардың ішінде ақпаратты өңдеу кейбір есептеулерді орындауды да қарастырады. Сондықтанда, компьютерге «түсінікті» санақ жүйесін қолдану қажеттілі туындады. Бұл сияқты мақсаттар үшін сандар тек 0 және 1 мәндерін қабылдай алатын екілік санақ жүйесі сай болды.

Жалпы алғанда, компьютер жадысында сандарды өңдеген уақытта бейнелеудің позициялық (орындық, позиционная) формасы қолданылады – сан (0 немесе 1) сандарының тізбегі ретінде беріледі, олардың әрқайсысының өз орынының санақ жүйесінің негізіне бөлінетін үлесі бар. Мысалы, 1001 екілік санына бейнелеудің позициялық формасындағы $1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$ жазбасы сәйкес келеді. Егер есептеулерді орындасақ 9 санын аламыз.

Сандардың позициялық формасы нақты сандарды жазуда да қолданылады, олардың бөлшек жақтарының ретінің мәні теріс болады. Мысалы, 45,36 ондық саны позициялық формада $4*10^1 + 5*10^0 + 3*10^{-1} + 6*10^{-2}$ түрінде жазылады. Сәйкесіше, нақыт сандар екілік санақ жүйесінде де бейнеленіледі.

Сонымен, екілік санақ жүйесі сандарды компьютерге «ұғынықты» формада бейнелеуге мүмкіндік береді.

Екілік сандарды экранда бейнелеу көп орынды талап етеді, сондықтан екіге бөлінетін санақ жүйелерін – сегіздік немесе он алтылық санақ жүйелерін қолдану керек деп шешілген.

Сандарды сегіздік жүйеде жазу үшін 0–ден 7–ге дейінгі 8 сан қажет. Үш екілік санның тізбегінің сегіз түрлі комбинациясы бар болғандықтан, екілік сандардың әр үйлесімін (триадасын) сегіздік санның біреуінің көмегімен бейнелеуге болады:

000 – 0
001 – 1
010 – 2
011 – 3
100 – 4
101 – 5
110 – 6
111 – 7

Мысалы, 01011110 екілік санын ыңғайлы болу үшін 01 011 110 триадаларының көмегімен жаамыз да, сегіздік санақ жүйесінде бейнелейміз 136, ал ол болса ондық жүйеде $1*8^2 + 3*8^1 + 6*8^0 = 94$ тең.

Санды он алтылық жүйеде жазу үшін 0–ден 9–ге дейінгі сандар және латын әліпбиінің A, B, C, D, E, F әріптері 16 сан қажет. Төрт екілік санның тізбегінің 16 түрлі комбинациясы бар болғандықтан, екілік сандардың әр үйлесімін (тетрадасын) ол алтылық санның біреуінің көмегімен бейнелеуге болады:

0000 – 0	1000 – 8
0001 – 1	1001 – 9
0010 – 2	1010 – A
0011 – 3	1011 – B
0100 – 4	1100 – C
0101 – 5	1101 – D
0110 – 6	1110 – E
0111 – 7	1111 – F

Мысалы, 01011110 санын ыңғайлы болу үшін 0101 1110 татрадаларының көмегімен жазамыз және оны алтылық жүйеде бейнелейміз 5E, ал ол болса ондық санақ жүйесінде $5*16^1 + 14*16^0 = 94$ тең.

Бағдарламалаушылар компьютерлік техникамен жұмыс жасағанда көбінесе, он алтылық санақ жүйесін қолданады, өйткені 1 байт ақпаратты екі он алтылық санмен бейнелеген оңай.

C# тілінде санды екілік немесе сегіздік санақ жүйесінде көрсету формалары жоқ, бірақ он алтылық санақ жүйесі кеңінен берілген.

Ондық санақ жүйесінің 0-ден 217-ге дейінгі сандарын он алтылық санақ жүйесіне түрлендірудің бағдарламалық жүзеге асырылуы берілген.

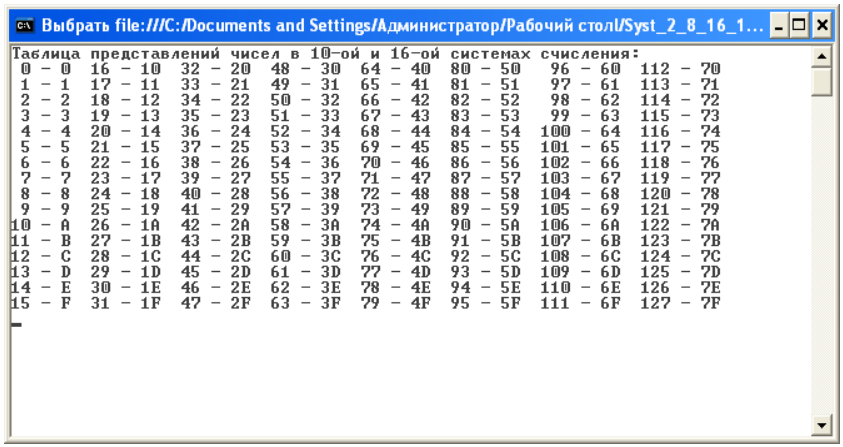
Бағдарламаның бастапқы коды:

```
using System;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
```

```
            uint i;
            Console.WriteLine("Таблица представлений чисел в 10-ой и
16-ой системах счисления:");
            for (i = 0; i <= 15; i++)
             Console.WriteLine("{0,2} - {1:X}  {2} - {3:X}  {4} -
{5:X}
               {6} - {7:X}  {8} - {9:X}  {10} - {11:X}  {12,3} -
{13:X}
               {14,3} -
{15:X}",i,i,i+16,i+16,i+32,i+32,i+48,i+48,i+64,
          i+64,i+80,i+80,i+96,i+96,i+112,i+112);
            Console.ReadLine();
           }
          }
         }
```

Бағдарлама жұмысы:



Сурет 2.1 – Ондық сандарды он алтылық сандарға түрлендіру

Егер екілік сандардың бейнесін «таза» түрде алу керек болса, онда әдетте ол үшін сандарды бейнелеудің «бағдарламалық» түрде толтырылатын жолдық формасы қолданылады, мысалы:

```
using System;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
   uint i;
   string d;
```

```
            char[] b = new char[8]
{'0','0','0','0','0','0','0','0'};
            Console.WriteLine("Таблица представлений чисел в 10-ой,
2-ой и 16-ой системах счисления:");
            for (i = 0; i <= 63; i++)
            {
             d = "";
             for (int j = 0; j < 8; j++) d = d + b[j];
             Console.WriteLine("{0,2} - {1} - {2:X}  ", i, d, i);
             if (b[7] == '0') b[7] = '1';
              else {b[7] = '0'; if (b[6] == '0') b[6] = '1';
               else {b[6] = '0'; if (b[5] == '0') b[5] = '1';
                else {b[5] = '0'; if (b[4] == '0') b[4] = '1';
                 else {b[4] = '0'; if (b[3] == '0') b[3] = '1';
                  else {b[3] = '0'; if (b[2] == '0') b[2] = '1';
                   else {b[2] = '0'; if (b[1] == '0') b[1] = '1';
                    else {b[1] = '0'; if (b[0] == '0') b[0] = '1';
                     else { b[0] = '0'; }}}}}}}}}
            Console.ReadLine();
          }
         }
        }
```

Таблица представлений чисел в 10-ой, 2-ой и 16-ой системах счисления:

0 - 00000000 – 0	17 - 00010001 – 11	34 - 00100010 – 22	51 - 00110011 - 33
1 - 00000001 – 1	18 - 00010010 – 12	35 - 00100011 – 23	52 - 00110100 - 34
2 - 00000010 – 2	19 - 00010011 – 13	36 - 00100100 – 24	53 - 00110101 - 35
3 - 00000011 – 3	20 - 00010100 – 14	37 - 00100101 – 25	54 - 00110110 - 36
4 - 00000100 – 4	21 - 00010101 – 15	38 - 00100110 – 26	55 - 00110111 - 37
5 - 00000101 – 5	22 - 00010110 – 16	39 - 00100111 – 27	56 - 00111000 - 38
6 - 00000110 – 6	23 - 00010111 – 17	40 - 00101000 – 28	57 - 00111001 - 39
7 - 00000111 – 7	24 - 00011000 – 18	41 - 00101001 – 29	58 - 00111010 - 3A
8 - 00001000 – 8	25 - 00011001 – 19	42 - 00101010 - 2A	59 - 00111011 - 3B
9 – 00001001 – 9	26 - 00011010 - 1A	43 - 00101011 - 2B	60 - 00111100 - 3C
10 - 00001010 – A	27 - 00011011 - 1B	44 - 00101100 - 2C	61 - 00111101 - 3D
11 - 00001011 – B	28 - 00011100 - 1C	45 - 00101101 - 2D	62 - 00111110 - 3E
12 - 00001100 – C	29 - 00011101 - 1D	46 - 00101110 - 2E	63 - 00111111 - 3F
13 - 00001101 – D	30 - 00011110 - 1E	47 - 00101111 - 2F	
14 - 00001110 – E	31 - 00011111 - 1F	48 - 00110000 - 30	
15 - 00001111 – F	32 - 00100000 – 20	49 - 00110001 - 31	
16 - 00010000 – 10	33 - 00100001 – 21	50 - 00110010 - 32	

Бағдарлама шынайы кестенің барлық мәндерін бір ұзын бағанға шығарады, бірақ дәрісте орынды үнемдеу үшін кесте 4 бағанмен берілген.

2.1.2 C# тілінің разрядтық логикалық операциялары

Разрядтық логикалық операциялары разрядтық логикалық көбейту немесе конъюнкция, разрядтық логикалық қосу немесе дизъюнкция және разрядтық логикалық шығару немесе НЕМЕСЕ-ні шығару операцияларымен берілген. Сонымен қатар, логикалық операцияларға разрядтық логикалық терістеу немесе толықтыру операциясы да жатады.

Екі санмен разрядтық логикалық көбейту операциясын (& белгіленеді) орындағанда олардың биттік (побитовое) «логикалық көбейтуі» орындалады да, нәтиже

сәйкес битке жазылады. Әрине, логикалық көбейтудің ережесіне сәйкес егер операцияға қатысатын екі битте 1-ге тең болса, нәтиже 1-ге тең болады.

Екі санмен разрядтық логикалық қосу операциясын (| белгіленеді) орындағанда олардың биттік «логикалық қосылуы» орындалады да, нәтиже сәйкес битке жазылады. Әрине, логикалық қосудың ережесіне сәйкес егер операцияға қатысатын екі биттің тым болмағанда біреуі 1-ге тең болса, нәтиже 1-ге тең болады.

Екі санмен разрядтық логикалық шығару, НЕМЕСЕні шығаратын операциясын (^ белгіленеді) орындағанда олардың биттік салыстырылуы орындалады да, және операцияға қатысатын екі биттің тек біреуі ғана 1-ге тең болса, нәтиже 1-ге тең болады.

Разрядтық логикалық терістеу, немесе толықтыру операциясын (~ белгіленеді) орындағанда операцияға қатысатын санның барлық разрядында бірлер нөлдермен, ал нөлдер бірлермен алмастырылады.

Материалды бекітіп алу үшін оқу бағдарламасының жұмысын қарастырайық, мұнда аталған барлық разрядтық логикалық операцияларды қолданамыз.

```
using System;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main()
  {
   uint a, b, c;
   Console.Write("Введите целое значение a ");
   a = Convert.ToUInt32(Console.ReadLine());
   Console.Write("Введите целое значение b ");
   b = Convert.ToUInt32(Console.ReadLine());
   c = a & b;
   Console.WriteLine("a  &  b = {0}   & {1}  = {2}", a, b,
c);
   Console.WriteLine("ax & bx = {0:X} & {1:X}= {2:X}", a,
b, c);
   c = a | b;
   Console.WriteLine("a  |  b = {0}   | {1}  = {2}", a, b,
c);
   Console.WriteLine("ax | bx = {0:X} | {1:X}= {2:X}", a,
b, c);
   c = a ^ b;
   Console.WriteLine("a  ^  b = {0}   ^ {1}  = {2}", a, b,
c);
   Console.WriteLine("ax ^ bx = {0:X} ^ {1:X}= {2:X}", a,
b, c);
   Console.WriteLine("~a  = {0}    ~b  = {1}", ~a, ~b);
   Console.WriteLine("~ax = {0:X}  ~bx = {1:X}", ~a, ~b);
   Console.ReadKey();
  }
 }
}
```

Бағдарлама жұмысы:
Введите целое значение a 42
Введите целое значение b 23

a & b = 42 & 23 = 2
ax & bx = 2A & 17 = 2
a | b = 42 | 23 = 63
ax | bx = 2A | 17 = 3F
a ^ b = 42 ^ 23 = 61
ax ^ bx = 2A ^ 17 = 3D
~a = 4294967253 ~b = 4294967272
~ax = FFFFFFD5 ~bx = FFFFFFE8

Разрядтық логикалық көбейту операциясының жұмысын тексерейік, a және b сандарының мәндерін екілік санақ жүйесінде бейнелейік:
ax = 2A = 0010 1010 – мәндер дәрістің алдыңғы пунктінің бағдарлама
bx = 17 = 0001 0111 жұмысының нәтижесінен алынған
ax & bx = 0000 0010 – нәтиже ондық (және он алтылық) 2 санына тең.

2.1.3 C# тіліндегі биттік жылжыту операциялары

Сандарды биттік жылжыту операциялары түрлі алгоритмдерде қолданылады, мысалы биттер тізбегін параллельді кодқа түрлендіруде және керісінше параллельді кодты биттер тізбегіне түрлендіруде қолданылады. Компьютердің кейбір құрылғылары бұл алгоритмдерді арнайы жылжыту регистрлерінің көмегімен аппаратты түрді жүзеге асырады, мысалы, магниттік немесе оптикалық дискілерден ақпаратты оқу және жазу сұлбасы (схема). Кейбір жағдайларда бұл алгоритм бағдарамалық түрде жүзеге асырылады, мысалы, USB арқылы қандай да бір мәліметтерді жіберу хаттамасын (протокол передачи данных) бағдарламалық түрде жүзеге асыру кезінде.

Екілік санды 2-ге көбейту арифметикалық операциясы екілік санды сол жаққа қарай бір разрядқа жылжытқанға пара пар , ал 2-ге бөлу операциясы – екілік санды бір разрядқа оңға жылжытқанға пара пар болады.

C# тілінде жылжыту операцияларын бағдарламалық түрде жүзеге асыру үшін екі операция бар:

- Екілік санды солға жылжыту операциясы (<< белгіленеді);
- Екілік санды оңға жылжыту операциясы (>> белгіленеді).

Екі операцияның да жазылу формалары бір-біріне ұқсас, операция белгісіне дейін жалжыту жасалынатын сан көрсетіледі, ал операциядан соң жылжыту жасалынатын разрядтар санын көрсеті керек, мысалы:

```
c = a << 2;                    // a санын солға екі разрядқа жылжыту.
c = b >> 1;                    // b санын оңға бір разрядқа жылжыту.
```

Жылжыту операциялары тек бүтін типті сандармен жұмыс істеуге арналған, мысалы, int, uint және т.б.

Солға жылжытқанда босаған разрядтар нөлге айналады.

Оңға жылжытқанда босаған разрядтар нөлге айналады, бірақ жылжыту санның таңбасын сақтай отыра жүргізіледі.

Мысал ретінде он алтылық 5 санын (екілік 0000 0000 0101 саны) 8 разрядқа солға жылжытуды қарастырамыз – нәтижесінде екілік 0101 0000 0000 саны немесе он алтылық 500 саны шығу керек.

Бағдарлама коды:

```
using System;

namespace ConsoleApplication1
{
 class Program
```

```
{
 static void Main()
 {
  uint a;
  Console.Write("Введите целое значение a ");
  a = Convert.ToUInt32(Console.ReadLine());
  for (int i = 1; i <= 8; i++)
  {
   a = a << 1;
   Console.WriteLine("ax = {0:X}      {1}", a, a);
  }
  for (int i = 4; i > 0; i--)
  {
   a = a >> 2;
   Console.WriteLine("ax = {0:X}      {1}", a, a);
  }
  Console.ReadKey();
 }
}
}
```

Бағдарлама жұмысы:

```
Введите целое значение a 5
ax = A     10
ax = 14     20
ax = 28     40
ax = 50     80
ax = A0     160
ax = 140     320
ax = 280     640
ax = 500     1280
ax = 140     320
ax = 50     80
ax = 14     20
ax = 5     5
```

a санын солға жылжыту циклда жылжыту қадамы бірге тең етіп жүргізіледі. Циклдің жұмыс нәтижесі санның әр қадамнан соң 5-тен 1280-ге дейін екі еседен өсіп отырғанын көрсетеді.

Санды оңға жылжыту да циклда жүргізілді, ал жылжыту қадамы 2 деп берілді. Бұл циклдің жұмыс нәтижесі a санының әр қадамнан соң 4 есеге азайып отырғанын көрсетеді.

2.2 C# тілінде қауіпсіз емес бағдарламалау

2.2.1 Windows жүйесінде идентификаторларды белгілеу

Windows жүйесінің функцияларымен жұмыс істеуші барлық жүйелік бағдарламалаушылар жүйеде қабылданған (венгер жазбалары деп аталатын) идентификаторлардың белгілеуін түсінуге тиіс. Осы белгілерді Microsoft-тың ең жақсы бағдарламалаушыларының бірі Чарльз Симонии (Венгрияда туған) қолдана бастады. Қылжақтап осы белгілерді венгер жазбалары деп атала басталды және барлық ізбасарлар мен

пайдаланушыларға (Windows-та бағдарламаларды әзірлейтіндерге) Windows-та қабылданған белгілердің ерекшеліктерімен танысу қажет болады. Идеяның мәні айнымалы, типтер және т.б. атаулары үшін олардың типін және бар ақпараттың сипатын суреттейтін префикстер қолданылатындығында. Кейбір префикстер келесі тізімде көрсетілген:

Префикс	Мәні
a	– Массив;
в	– Логикалық тип (BOOL);
by	– Таңбасы символдық тип (BYTE);
c	– Символдық тип;
cb	– Байттардың санауышы;
cr	– Түс;
cx,cy	– Қысқа тип (SHORT);
dw	– Таңбасы жоқ ұзын тип (DWORD);
fn	– Функция;
h	– Логикалық нөмер (HANDLE немесе HINSTANCE);
i	– Бүтін;
m_	– Клас айнымалысы;
n	– SHORT немесе int;
np	– жақын нұсқағыш;
p	–Нұсқағыш;
l	– Ұзын тип (LONG);
lp	– Алыс нұсқағыш;
s	– Жол;
cz	– Нөль-символымен аяқталатын жол;
w	– Таңбасы жоқ бүтін (WORD);
x,y	– Қысқа тип(х немесе у координаты).

Мысалы, lpfnWndProc жазбасы lpfnWndProc-тің WndProc функциясына алыс нұсқағыш болатынын көрсетеді.

Келтірілген мысалдарда нұсқағыш түсінігі бірнеше рет ескерілді. Нұсқағыштарды пайдаланумен бағдарламалау технологиясы Windows жүйесін жазу кезінде кеңінен қолданылды. Сондықтанда бізге нұсқағыш түсінігін және онымен жұмыс істеу ережелерін ұғыну қажет. Бұл өте маңызды, өйткені біз қолданатын Windows жүйесінің API функцияларының көптеген формалды параметрлері нұсқағыштар болып табылады.

«Нұсқағыштар типтері негізінен С тілінің API-интерфейстерімен әрекеттескенде пайдалы болады, бірақ оларды басқарылатын шоғыр (куча) шегінен тыс болатын жадыға жүгіну үшін немесе бағдарламаның жоғары өнімді бөлімдерін жүзеге асырғанда пайдалануға болады» [Джозеф Албахари б. 171. C# 3.0 анықтамалығы]

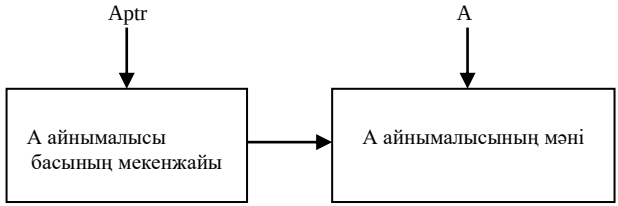
2.2.2 Нұсқағыш түсінігі

Нұсқағыштар – бұл өз мәндері ретінде компьютер жадысының мекенжайын ұстайтын айнымалылар.

Осы тұрғыда айнымалының анықтамасын ескерту керет. Айнымалы – идентификатормен (айнымалы атауымен) белгіленген жады облысы, онда бағдарлама жұмысының барысында өзгерілетін деректер сақталынуы мүмкін. Айнымалы атауында осы айнымалының деректері орналасқан жадының мекенжайы жөнінде ақпарат болуға тиіс (әйтпесе біз өз деректерімізді компьютер жадысынан таппаймыз). Бірақ, қарапайым айнымалының мәні тұрған компьютер жадысының физикалық мекенжайы пайдаланушыға қол жетімсіз болса, ал айнымалы нұсқағыш компьютер жадысының физикалық

мекенжайларымен, оның ішінде қарапайым айнымалы деректері тұрған жадының мекенжайларымен жұмыс істеу үшін арналған.

Яғни, егерде А айнымалы болса, А айнымалысының деректері бар жады мекенжайы орналасқан Aptr айнымалысы бар деп елестетуге болады. А айнымалысымен Aptr айнымалы-нұсқағышы арақатынасын келесі суретпен көрсетуге болады:



Сурет 3.1 – А айнымалысы және оның Aptr нұсқағышы

Нұсқағыштар кез келген басқа айнымалыларға ұқсас бағдарламадаға өзінің қолданылуының алдында хабардар етілуге тиіс. Нұсқағыштың хабардар етілу (сипатталу) форматы келесі түрде болады:

тип* айнымалы; мұнда

тип – мекенжайы айнымалыда сақталынатын жады облысындағы мәннің типі. Тип клас бола алмайды, бірақ құрылым, тізбелеу немесе нұсқағыш, сондай-ақ мәнді типтердің кез келгені және void бола алады. Void типі нұсқағыш белгісіз типті айнымалының мекенжайын сақтайтынын білдіреді. Нұсқағыштады сипаттаудың мысалдары:

int* aptr;

int a;

біз бұл мысалда екі айнымалыны сипаттаймыз: а – бүтін типті қарапайым айнымалы; aptr –бүтін типті мәнге нұсқағыш.

float* xptr, yptr;

бұл жағдайда xptr және yptr айнымалылары нақты типті мәндерге нұсқағыштар болып табылады.

int*[] mas;

мысалда бүтін типті мәндерге нұсқағыштар массивін сипаттау нұсқасы қарастырылған.

int** iptr;

мысалда бүтін типті нұсқағышқа нұсқағыш сипатталған.

Айнымалыларды сипаттауда «*» символы оған сәйкес келетін айнымалы нұсқағыш болатынын көрсетеді (яғни мекенжайды сақтауға арналған). Нұсқағыштың мәні – бұл компьютер жадысының мекенжайы екенін тағы да ескертеміз.

2.2.3 Нұсқағыштың инициализациялау операциясы

Нұсқағыш бұл айнымалы, сондықтан да онымен кейбір операциялар жасауға болады.

Кез келген айнымалының басты операциясы инициализациялау операциясы (тағайындау) – айнымалыға қандайда бір мән тағайындалады. Нұсқағыш ерекше емес, бірақ мән ретінде компьютер жадысының мекенжайы алынуы тиіс. C# тілінде компьютер жадысының физикалық мекенжайын алу үшін арнайы операция бар – «&» адрестеу

операциясы, ол өзі алдынан орнатылған айнымалының немесе басқа деректердің жадысының мекенжайын қайтарады. Мысалы,

Aptr = &a;

Осы мысалда Aptr нұсқағышына a айнымалысының мекенжайы тағайындалатын болады – a айнымалысының деректері сақталынатын компьютер жадысы облысының бастапқы мекенжайы. Мұнда Aptr a айнымалысы сияқты типтегі айнымалы нұсқағышы ретінде сипатталынуы тиіс.

Нұсқағыш мәніне басқа нұсқағыштың мәнін тағайындауға болады, бірақ тек типтері бірдей болу керек, мысалы, егерде Aptr және Bptr бүтін типті мәндерге нұсқағыштар болса және Bptr мәні анықталған болса (әйтпесе тағайындаудың мағынасы жоқ), онда келесі жазудың мағынасы бар болады:

Aptr = Bptr; .

C# тіліндегі айнымалыларды сипаттау олардың инициализациялауымен бірге жүруі тиіс, сондықтанда C# тілінде нұсқағыштар үшін компьютер жадысының жалған «нөлдік» мекенжайының константасы енгізілген - null. Кез келген типтегі нұсқағыштың мәні инициализациялау кезіңінде де, бағдарлама жұмысының барысында да компьютер жадысының осы «нөлдік» мекенжайына меншіктелінуі мүмкін, мысалы:

```
int* Aptr = null;
float Fptr;
Fptr = null;
```

Осы мысалда Aptr және Bptr нұсқағыштарға null деген компьютер жадысының жалған «нөлдік» мекенжайы тағайындалған.

Бағдарлама жұмысының барысында (немесе нұсқағыштардың инициализациясы кезінде) нұсқағыштарға айқын түрінде компьютер жадысының мекенжайын тағайындауға болады, бірақ бұл тағайындау тек осы мекенжай анық белгілі болса ғана жүргізіледі, өйткені олай болмаған жағдайда бағдарлама жұмысынан жаңылуы мүмкін, мысалы:

```
byte* Aptr = (byte *) 0x3F0ECD4;
char* Cptr = (char *) 0x3F0ECD4;
```

мұнда, 0x3F0ECD4 – жады мекенжайының он алтылық константасы, (byte *) немесе (char *) – константа нұсқағыштың сәйкес типіне келтірілетін типтерді келтіру операциялары.

Нұсқағыштар массивын инициализациялау және нұсқағыштардың көмегімен стекке жадыны бөлу мысалдарын нақты мысалдарда кейінірек қарастырамыз.

2.2.4 Нұсқағыштармен басқа операциялар

C# тілінде «*» символымен белгіленетін мәліметтер элементіне қол жеткізу /разыменование/ операциясы болады. Бұл операция адрестеу операциясына кері операция. Егерде адрестеу операциясы деректер бойынша олардың мекенжайын алса, ал мәліметтер элементіне қол жеткізу /разыменование/ операциясы мекенжайы бойынша деректерді алады. әрине, осы операцияға нұсқағыш қатысу керек. Мысалы,

a=*Aptr;

а айнымалысына мекенжайы Aptr нұсқағышында болатын деректер тағайындалады. Әрі Aptr а айнымалысы сияқты типтегі айнымалының нұсқағышы ретінде сипатталынуы тиіс.

C# тілінде құрылым типіндегі деректермен жұмыс кезінде «элементке нұсқағыш» деген арнайы операция немесе «->» символдарының комбинациясымен белгіленетін нұсқағыш арқылы құрылым элементіне қатынау операциясы бар. Мысалы, егерде PtrCtyd қандай да бір құрылымға нұсқағышты білдірсе, ал Name осы құрылымның өрісі болса, сонда PtrCtyd->Name жазбасы (*PtrCtyd).Name эквивалентті. Яғни *PtrCtyd мекенжайы бойынша біз бүкіл құрылымға жүгінеміз және онда Name өрісін таңдаймыз.

Нұсқағыштармен орындалатын негізгі операциялар 3.1-кестеде берілген.

3.1-кесте. Нұсқағыштармен орындалатын негізгі операциялар

Операция	Сипаттамасы
*	мәліметтер элементіне қол жеткізу /разыменование/ операциясы-нұсқағышта сақталған адресте орналасқан мәнді алу
->	Нұсқағыш арқылы құрылым элементіне қол жеткізу
[]	Нұсқағыш арқылы массив элементіне қол жеткізу
&	Айнымалы мекенжайын алу
++, --	Нұсқағыштың мәнін бір адрестелінетін элементке ұлғайту және азайту
+, -	Нұсқағыштарды бүтін шамалармен қосу және алу
==, !=, <>, <=, >=	Нұсқағышта сақталынған мекенжайларды салыстыру. Таңбасы жоқ бүтін шамалардың салыстыруы сияқты орындалады
Stackalloc	Стекте нұсқағыш сілтеп тұрған айнымалы үшін жады бөлу

2.2.5 Қауіпсіз емес код түсінігі

C# тілінің негізгі артықшылықтарының бірі – оның жадымен жұмыс істеу сызбасы: объектілер үшін жадыны автоматты түрде бөлу және қоқысты автоматты түрде тазалау. Әрі жадының қолданыста жоқ мекенжайына жүгінуге немесе массивтың шекараларынан асуға болмайды, бұл бағдарламаларды неғұрлым сенімді және қауіпсіз етеді және бір қатар кластардың қателерінің болуына жол бермейді.

Нұсқағыштар тікелей жады облысының мекенжайларымен жұмыс істеуге мүмкіндік береді, ал бұл C# тілінің қағидаттарына қарама-қайшы болады. Сондықтанда нұсқағыштарды қолданатын бағдарламалық код қауіпсіз емес деп аталатын болды.

Жалпы жағдайда CLR ортасы орындалуын бақыламайтын код қауіпсіз емес код деп аталынады.

CLR ортасымен дауды шешу үшін осындай код анық түрде unsafe негізгі сөзінің көмегімен белгіленуі керек. Осы сөз орындаудың қауіпсіз емес контекстін анықтайды.

Unsafe деген негізгі сөз класты, құрылымды немесе құрылым деректерінің өрісін сипаттау кезінде сипаттаушы ретінде қолданылуы мүмкін, мысалы:

public unsafe struct Ctyd {}.

Бұл жағдайда құрылымның барлық өрістері қауіпсіз емес ретінде белгіленеді. Егерде unsafe негізгі сөзі құрылымның тек бір өірісін сипаттау үшін сипаттаушы ретінде қолданылса, мысалы:

public struct Ctyd {… public unsafe int * Kol; … }

онда қауіпсіздік емес шарты осы құрылымның тек белгіленген өрісіне ғана таратылады.

unsafe негізгі сөзі күрделі оператор ретінде қолданылуы мүмкін, мысалы:

unsafe { блок }, мұнда белгіленген блокқа енген барлық операторлар қауіпсіз емес болады - яғни олардың жұмысы CLR ортасымен бақыланбайды және қауіпсіз емес айнымалыға бөлінген динамикалық жады C# тілінің қоқыс жинаушысымен «қорғалынбайды».

2.2.6 Нұсқағыштармен жұмыс бағдарламасының мысалы

Нұсқағыштармен жұмыс жасау ережесін жақсы түсіну және олармен операцияларды орындау үшін таза оқу мысалын қарастырамыз. Осы мысалда адрестеу және мәліметтер элементіне қол жеткізу /разыменование/ операцияларының жұмысын қарастырамыз.

Қауіпсіз емес фрагменті бар консолдық қосымша кодының жемісті компиляциясы үшін келесі жолмен Visual Studio ортасының баптауын өзгерту қажет: Project->

ConsoleAplication1 Properties-> Build командаларын таңдап, ашылған терезеде Allow Unsafe Code пунктінде «қанат белгі» («***птичка***») орнату керек. Тереземен жұмысты Advanced батырмасы арқылы аяқтап, өзгертулерді OK батырмасын басумен растаймыз.

Қосымша кодында unsafe негізгі сөзін Main әдісінің сипаттауышы ретінде қолдануға болады, мысалы, `static unsafe void Main()`, сонда осы әдістің барлық операторлары қауіпсіз емес ретінде қарастырылатын болады.

Бағдарламаның бастапқы коды (кодтың кейбір фрагменттері Павловскаяның кітабынан алынған):

```
using System;

namespace ConsoleApplication1
{
 class Program
 {
  static unsafe void Main()
  {
   int a, c;
   uint adr;
   a = 10;
   int* aPtr = null;
   aPtr = &a;
   Console.WriteLine("*aPtr = " + *aPtr);
   Console.WriteLine("a = " + a);
   Console.Write("aPtr = ");
   adr = (uint)aPtr;
   byte* b = (byte*)&adr;
   b = b + 3;
   Console.Write("0x");
   for (int i = 0; i < 4; i++)
       { Console.Write("{0:X}", *b); b--; }
   Console.WriteLine();
   Console.Write("&a   = ");
   adr = (uint)&a;
   b = (byte*)&adr;
   b = b + 3;
   Console.Write("0x");
   for (int i = 0; i < 4; i++)
       { Console.Write("{0:X}", *b); b--; }
   Console.WriteLine();
   Console.WriteLine("adr = " + adr);
   c = *& a;
   Console.WriteLine("*& a = " + c);
   //c = &* a;
   //Console.WriteLine("&* a = " + c);
   Console.Write("&* aPtr = ");
   adr = (uint)&* aPtr;
   b = (byte*)&adr;
   b = b + 3;
   Console.Write("0x");
   for (int i = 0; i < 4; i++)
       { Console.Write("{0:X}", *b); b--; }
   Console.WriteLine();
```

```
            Console.Write("*& aPtr = ");
            adr = (uint)*& aPtr;
            b = (byte*)&adr;
            b = b + 3;
            Console.Write("0x");
            for (int i = 0; i < 4; i++)
                { Console.Write("{0:X}", *b); b--; }
            Console.WriteLine();
            Console.ReadKey();
          }
        }
    }
```

Бағдарлама жұмысы:

```
*aPtr = 10
a = 10
aPtr = 0x3F0ECD4
&a   = 0x3F0ECD4
adr = 66120916
*& a = 10
&* aPtr = 0x3F0ECD4
*& aPtr = 0x3F0ECD4
```

Мәліметтер элементіне қол жеткізу /разыменование/ және адрестеу сияқты екі қатар операциялары нұсқағыш үшін де, солай тұтас типті айнымалы үшін де өзара бір-бірін жояды (*& a = 10 қараңыз).

Мәліметтер элементіне қол жеткізу /разыменование/ және адрестеу сияқты екі қатар операциялары тек нұсқағыш үшін ғана өзара бір-бірін жояды. Тұтас типті айнымалы үшін осы комбинацияны қолдану әрекеті Error 1 The * or -> operator must be applied to a pointer деген хабарламамен бағдарлама компиляциясының қатесіне әкеледі.

Нұсқағыштармен жұмыс кезінде егерде нұсқағыштар бағдарламаның басында инициализацияланатын болса, бұл жақсы стиль деп саналады, яғни оларға немесе кейбір мәндер тағайындалса, немесе нұсқағыштарға null-ға тең нөлдік мәндер берілсе.

2.2.7 void* типі үшін нұсқағышты пайдалану

Нұсқағышты сипаттау кезінде void* типін пайдалануға болады, бұл кейінге қалдырған типті нұсқағышты сипаттауды білдіреді.

Осындай void* типіндегі нұсқағыш үшін арифметикалық операцияларды және мәліметтер элементіне қол жеткізу /разыменование/ операцияларын қолдануға болмайды.

Бірақ,ол void* типті нұсқағыштың кез келген типті нұсқағышқа анық емес түрлендірілуі үшін қолданылуы мүмкін. Сол уақытта void* типіндегі нұсқағышқа кез келген типтегі айнымалының мекенжайын тағайындауға болады. Сонымен, void* типіндегі нұсқағышты кез келген типті айнымалылардың мекенжайларының аралық сақтауышысы ретінде пайдалануға болады. Таза оқу мысалын қарастырамыз, онда void* типіндегі нұсқағышқа кезек-кезегімен бүтін және нақты типті айнымалыларының мекенжайларын тағайындайтын боламыз, бұдан соң осы мекенжайларды тиісінше бүтін және нақты типтер үшін нұсқағыштарға тапсырамыз.

```
using System;

namespace ConsoleApplication1
{
```

```
class Program
{
 static unsafe void Main()
 {
  int a = 10;
  double b = 15.67;
  int* aPtr = null;
  double* bPtr = null;
  void* cPtr=null;
  for (int i = 1; i <= 5; i++)
  {
   if (i % 2 == 0)
   {
    cPtr = &a; aPtr = (int*)cPtr;
    Console.WriteLine("i = {0} По адресу указателя
находится число {1} ", i, *aPtr);
   }
   else
   {
    cPtr = &b; bPtr = (double*)cPtr;
    Console.WriteLine("i = {0} По адресу указателя
находится число {1} ", i, *bPtr);
   }
  }
  Console.ReadKey();
 }
}
}
```

Бағдарлама жұмысы:
i = 1 По адресу указателя находится число 15,67
i = 2 По адресу указателя находится число 10
i = 3 По адресу указателя находится число 15,67
i = 4 По адресу указателя находится число 10
i = 5 По адресу указателя находится число 15,67

2.3 Массивтермен жұмыс жасағанда нұсқағыштарды қолдану

2.3.1 Деректер массивы бойынша нұсқағышты жылжыту үшін қосу және алу операцияларын қолдану

Нұсқағыштарға кейбір арифметикалық операциялар қолдалынуы мүмкін, бірақ оларды пайдалану деректерді компьютер жадысында орналастыру бойынша жақсы білімдерді қажет етеді. Өйткені нұсқағыштың бір бірлікке ұлғаюы нұсқағышқ жататын типті келесі деректің мекенжайына ауысуды білдіреді.

Қарапайымдылық үшін деректер ретінде **char** типіндегі айнымалылардың массивын қолданамыз. Оларға массивтың инициализациялауы кезінде латын әліпбиінің алғашқы 10 әріпін жазамыз. Массив бойынша нұсқағышты жылжытып, жеке әріптерді баса отырып, біз қандай да бір сөзді баса аламыз, мысалы, «ВЕНЕFIС».

Бағдарламаның бастапқы коды:

```
using System;
```

```
namespace ConsoleApplication1
{
 class Program
 {
  static unsafe void Main()
  {
   char[] masc = new char[]
                 {'A','B','C','D','E','F','G','H','I','J'};
   fixed (char* Mptr = masc)
   {
    char* ptr = Mptr;
    ptr++;
    Console.Write(*ptr);
    ptr = ptr +3;
    Console.Write(*ptr);
    ptr += 3;
    Console.Write(*ptr);
    ptr -= 3;
    Console.Write(*ptr);
    ptr++;
    Console.Write(*ptr);
    ptr += 3;
    Console.Write(*ptr);
    ptr -= 6;
    Console.Write(*ptr);
   }
   Console.WriteLine();
   Console.ReadKey();
  }
 }
}
```

Бағдарлама жұмысы:
BEHEFIC

Бағдарламада fixed операторы пайдаланылған, ол келесі жазылу форматына ие:
fixed ([type][*] [имя указателя] = [&][имя объекта])
{ тіркелген объектімен әрекеттер (действия с фиксированным объектом) }

«**fixed** операторы, қоқысты тазалаушы орнын ауыстырмауы (жылжытпауы) үшін мекенжайы нұсқағышқа енгізілетін объектіні бекітіп алады, осылай нұсқағыш дұрыс болып қалады. Бекіту дөңгелек жақшалардан соң жазылған блокты орындау уақытында өтеді.» [Павловская, б.352].

Мәселе қоқыс тазалаушы үнемі жұмыс істеуде болады және шоғыр жадыны оңтайландырумен шұғылданады - шоғыр жадысын оңтайландыру мақсатында жұмысын аяқтаған объектілерді жою және жұмыс істеп тұрған объектілерді жылжытумен шұғылданады.

Егерде біздің объект осындай оңтайландырудың нәтижесінде жадының басқа облысына ауысса, онда оның мекенжайлары өзгереді және ескі мекенжайлармен нұсқағыштарды пайдалану өз мәнін жояды. Іс жүзінде **fixed** операторы қоқыс тазалаушыға көрсетілген объектіні шоғырға жылжытуға болмайтындығын ескертеді.

Mptr нұсқағыштың көмегімен символдар массивының бастапқы мекенжайын бекітіп алып, біз массив элементтері тізімінің тақырыбын құрған боламыз. Массив

элементтерінің тізімі бойынша жылжу үшін тағы бір нұсқағыш – инициализациялау кезінде тақырыпқа орнатылатын **ptr** ағымдағы нұсқағышы қажет. **Fixed** операторы блогының ішінде тақырыпты өзгертуге жасалған қарекеттер бағдарламаның компиляциясының қатесіне әкеп соғады, өйткені деректердің бекітілген облысының тақырыбын өзгерту деректердің жоғалуына әкеп соғу мүмкін еді.

Нұсқағыштарға арифметикалық операцияларды пайдалану компьютер жадысындағы деректердің мекенжайларына сәйкес келетін бүтін мәндермен ғана әрекеттер жасауға рұқсат береді. Мұнда нұсқағыштармен адресталатын деректер типінің көлемі автоматты түрде реттеледі. Элементтерінің мәндері компьютер жадысында реттімен орналастырылған массивтармен жұмыс кезінде арифметикалық операцияларды пайдалану ұсынылады.

2.3.2 Жадыда деректерді орналастыру үшін stackalloc операциясын пайдалану

C# тілінде **new** операциясының көмегімен нұсқағыштардың деректеріне жадыны бөлу қолданылмайды. Нұсқағыштар көмегімен адресталатын деректерге жадыны бөлудің негізгі тәсілі **stackalloc** операциясы, әрі мұнда жады стекте бөлінеді.

stackalloc операциясын жазу форматы:

тип* нұсқағыш атауы= **stackalloc** тип [саны];

Оқу бағдарламасы ретінде тұтас типті 10 айнымалы үшін жадының бөлуді және оларды таңдау әдісімен сұрыптауды қарастырамыз.

```
using System;

namespace ConsoleApplication1
{
 class Program
 {
  static unsafe void Main()
  {
   int i, j, b;
   int* masptr = stackalloc int[10];
   // Проверка инициализации массива
   for (i = 0; i < 10; i++)
   {
    Console.Write(" {0}", masptr[i]);
   }
   Console.WriteLine();
   Random rnd = new Random();
   // формирование и печать массива
   Console.Write("Массив до сортировки:    ");
   for (i = 0; i < 10; i++)
   {
    masptr[i] = rnd.Next() % 101 - 50;
    Console.Write(" {0}", masptr[i]);
   }
   Console.WriteLine();
   // сортировка элементов массива методом выбора
   for (i = 0; i < 9; i++)
    for (j = i + 1; j < 10; j++)
     if (masptr[i] < masptr[j])
    { b = masptr[i]; masptr[i] = masptr[j]; masptr[j] = b; }
```

```
            // печать массива после сортировки
            Console.Write("Массив после сортировки: ");
            for (i = 0; i < 10; i++)
             Console.Write(" {0}", masptr[i]);
            Console.WriteLine();
            Console.ReadKey();
           }
          }
        }
```

Бағдарлама жұмысы:

0 0 0 0 0 0 0 0 0 0

Массив до сортировки: 23 -27 -35 33 -31 -31 23 4 42 40

Массив после сортировки: 42 40 33 23 23 4 -27 -31 -31 -35

Егерде жадыны бөлу үшін нұсқағыш пайдаланығанын білмесек, онда бағдарламада `masptr` деген қызық атаумен қарапайым массив пайдаланған деп болжауға болады.

Массивқа нұсқағыш арқылы жүгінген кезде орта массив шекарасының бұзылуын бақыламайды және де индексті тым өзгерте отырып, массив үшін бөлінген жадының шекарасынан «шығу» және кейбір деректерді немесе бағдарламаны бұзу қауіпі бар. Массивқа нұсқағыштың көмегімен жүгіну кезіндегі екінші ыңғайсыздық нұсқағыштармен жұмыс істемейтін Length қасиетінің көмегімен массив ұзындығын анықтай алмау болып табылады.

Стекте жадыны бөлу кезінде массивтың барлық элементтеріне нөлдік мәндер беріледі – деректердің алғашқы инициализациясы орындалады.

Stackalloc операциясының көмегімен жадыны жеке айнымалыларға бөліп беруге болады – бір элементті массив сияқты. Бағдарламаның келесі фрагментінде екі нұсқағыштың пайдаланылуы көрсетілген, олар үшін жады стекте **stackalloc** операциясының көмегімен бөлінген болатын.

```
int* Aptr = stackalloc int[1];
int* Bptr = stackalloc int[1];
. . .
Aptr[0] = 5;
Bptr[0] = Aptr[0];
Console.WriteLine("Bptr[0] =" + Bptr[0]);
```

Фрагмент жұмысының нітижесінде шығады: `Bptr[0] = 5`

Стекте жадыны босату **stackalloc** операциясы орналасқан блоктың жұмысы аяқталғанда автоматты түрде жүргізіледі – біздің жағдайымызда бұл бағдарлама.

2.3.3 Нұсқағыштар массивын пайдалану

Нұсқағыштар, қарпайым айнымалылар сияқты, массивта сақталынуы мүмкін. Нұсқағыштар массивын сипаттау келесі форматқа ие:

тип* [] нұсқағыштар _массивының_ атауы = new тип*[элементтер_саны];

Мысалы, int* [] masiPtr = new int*[10];

Нұсқағыштар массивын пайдаланудың оқу мысалы ретінде минус 50-ден 50-ге дейін диапазонында 10 кездейсоқ санды қалыптастырылуын, олардың жадыға, ал олардың мекенжайларының нұсқағыштар массивына жазылуын қарастырамыз.

Бағдарламаның бастапқы коды:

```
using System;
```

```
namespace ConsoleApplication1
{
 class Program
 {
  static unsafe void Main()
  {
   int i, j, b;
   int*[] masiPtr = new int*[10];
   Random rnd = new Random();
   // формирование массива
   for (i = 0; i < 10; i++)
   {
    int* aPtr = stackalloc int[1];
    aPtr[0] = rnd.Next() % 101 - 50;
    masiPtr[i] = &aPtr[0];
   }
   // печать массива
   Console.Write("Массив :    ");
   for (i = 0; i < 10; i++)
       Console.Write(" {0}", *masiPtr[i]);
   Console.WriteLine();
   Console.ReadKey();
  }
 }
}
```

Бағдарлама жұмысы:
Массив : -8 -43 31 19 23 31 -36 -46 -2 -21

Массив элементінің әр мәні үшін жадының бөлінуі `int* aPtr = stackalloc int[1];` операциясының көмегімен жүргізіледі, ал массив элементтерінің мәндері жадысының мекенжайлары нұсқағыштардың массивына жазылады.

2.3.4 Құрылымдармен жұмыс кезіндегі нұсқағыштарды пайдалану

Нұсқағыштардың көмегімен құрылымдармен жұмыс кезінде оқу мысалы ретінде жазбалар стегін құрамыз, ал қандай да бір бүтін типті айнымалы және стектің келесі элементіне нұсқағыш олардың өрістері болады (ұқсас жазбаға нұсқағыш).

```
using System;

namespace ConsoleApplication1
{
 class Program
 {
  unsafe struct zveno
  {
   int nom;
   zveno* next;
   static unsafe void Main()
   {
      zveno zap = new zveno();
      zveno* tek1, tek2;
```

```
    zveno* zagol;
    zagol = &zap;
    tek2 = zagol;
    zagol->nom = 10;
    zagol->next = null;
    for (int i = 1; i < 5; i++)
    {
     zveno* zap1 = stackalloc zveno[1];
     tek1 = &zap1[0];
     tek2->next = tek1;
     tek1->next = null;
     tek1->nom = i;
     tek2 = tek1;
    }
    tek1 = zagol;
    while (tek1 != null)
    {
     Console.Write("  " + tek1->nom);
     tek1 = tek1->next;
    }
    Console.WriteLine();
    Console.ReadKey();
   }
  }
 }
}
```

Бағдарлама жұмысы:

```
10        1 2 3 4
11
```

3 C# ТІЛІНДЕГІ ТҰРАҚТЫ ӨРНЕКТЕР

3.1 Тұрақты өрнектер түсінігі

Бағдарламалаудың кез келген тілінде мәтіндік ақпаратпен жұмыс істеу үшін арнайы құралдар бар. Әдетте, олар string, char типіндегі айнымалылар және осы айнымалылармен жұмыс істеуге арналған операторлар, функциялар немесе кластардың жиынтығы.

Тұрақты өрнектердің мәтіндік ақпаратты өңдеуге арналған бағдарламалаудың мамандандырылған тілі болып келеді.

.NET кітапханасының тұрақты өрнектері Perl5 тілінің тұрақты өрнектеріне негізделген және де өзіне оның функционалдық мүмкіншіліктерінің үлкен бөлігін енгізеді.

Әрине, тұрақты өрнектермен есептелініп шешілетін кез келген есеп System.String және System.Text.StringBuilder кластары әдістерінің көмегімен жүзеге асырылуы мүмкін, бірақ осы шешімдердің кодтарының көлемдері тұрақты өрнектердің көмегімен жазылған код көлемінен бірнеше есе үлкен болады.

Жүйелік бағдарламалаушылармен тұрақты өрнектердің кейбір элементтерін операциялық жүйлермен «командалық жол» режімінде жұмыс істегенде жемісті қолданылады.

3.2 Тұрақты өрнектермен шешілетін негізгі міндеттер

.NET кітапханасының тұрақты өрнектері мәтіндік ақпаратпен жұмыс істеу кезінде келесі негізгі міндеттердің шешімін қамтамасыз етеді:

мәтінде берілген фрагменттің болуын тексеру;

мәтінді енгізуге қол жетімділігін тексеру, мысалы, парольдерді;

мәтінде берілген шаблон бойынша іздестіру;

мәтін фрагменттерін редакциялау, ауыстыру және жою;

кез келген құрылымды файлдарынан деректерді алу, мысалы, HTML-беттері;

мәтінмен жұмыс нәтижелері бойынша қорытынды есеп берулерді қалыптастыру.

Әрине, бұл тұрақты өрнектердің көмегімен шешілетін міндеттердің бір бөлігі ғана. Олардың кейбіреулерін біздің дәрістерімізде қарастыратын боламыз.

Сонда C# тіліндегі жолдар өзгерілмейтін объектілер болатынын естен шығармау керек, сонымен тұрақты өрнектер бастапқы жолды өзгерте алмайды.

3.3 Тұрақты өрнектер тілінің символикасы

Тұрақты өрнектер тілінің символикасы қарапайым және метасимволдар сияқты екі түрлі символдармен берілген.

Мәтінде қарапайым символдар өздерін өздері көрсетеді, мысалы, мәтінде іздестіру кезінде пайдаланылатын «Само*» шаблоны, осы шаблон «Сам» символымен басталатын сөздерді іздейтін болады. Мысалы, «Сам», «Само» немесе «Самоо». «*» символы метасимвол болып табылады және алдыңғы «о» символы 0 немесе одан да көп рет қайталануы мүмкін екенін білдіреді. Сонымен, қарапайым символдар мәтіндік ақпаратты жазуға мүмкіндік береді.

Метасимволдарда символдық мән емес арнайы мән бар.

Әрине, тұрақты өрнектер тілінде «резервтелген» символдар бар, олардың пайдаланылуы тілмен символ сияқты емес метасимвол ретінде қабылданады. Мысалы, алдыңғы мысалдағы «*» символы.

Егерде мәтінде * метасимволын емес «*» символын пайдалану қажет болса, онда оны пайдалану алдында «\» символын орнату қажет, мысалы, 3*a – 3-ті a-ға көбейту.

Егерде әріпті метасимвол ретінде пайдалану қажет болса, онда оны пайдалану алдына «\» символын орнату қажет, мысалы, «Lab\d» жазбасы мәтінде «\d» орнына сияқты кез келген ондық цифр бола алатынын білдіреді - Lab1 немесе Lab6.

Метасимволдар өте көп. Әдетте, әдебиетте метасимволдар олардың функционалдық тағайындалуларына байланысты топтармен қарастырылады. Мысалы, анықтаушы метасимволдар, қайталағыштар немесе квантификаторлар, ауыстырғыштар немесе «Символдар кластары» және т.б. Олардың нақты зерделенуі үшін әдетте дәрістер емес анықтамалық әдебиет пайдаланылады. Бірақ, көптеген студенттер «анықтамалық әдебиетті оқи алмайды» және оқытушылар дәрістерінде дәстүрлі түрде тұрақты өрнектер тілінің негізгі метасимволдарын келтіруге мәжбүрлі. Дәстүрлерден шегінбестен, олардың функционалдық бағытталуына байланысты кейбір метасимволдарды қарастырамыз.

3.4 Қайталағыштар

Қайталағыш метасимволдары (әдебиетте оларды кейде квантификаторлар деп атайды) қарапайым символдан немесе символдар класынан кейін орналасады және мәтіндегі мүмкін болатын қайталаулардың санын анықтайды.

3.1 кестесінде келтірілген ең жиі пайдаланылатын қайталағыштардың қарастырамыз.

3.1-кесте

Қайталағыштар

Метасимвол	Сипаттамасы	Мысал
*	Алдыңғы элементтің нөл немесе одан да көп рет қайталануы	Выражение ca*t соответствует фрагментам ct, cat, caat, caaaaaaaaaat и т. д.
+	Алдыңғы элементтің бір немесе одан да көп рет қайталануы	Выражение ca+t соответствует фрагментам cat, caat, caaaaaaaaaat и т. д.
?	Алдыңғы элементтің нөл немесе бір рет қайталануы	Выражение ca?t соответствует фрагментам ct и cat
{n}	Алдыңғы элементтің тура n рет қайталануы	Выражение ca{3}t соответствует фрагменту caaat, а выражение (cat){2} — фрагменту catcat
{n,}	Алдыңғы элементтің тым болмағанда n рет қайталануы	Выражение ca{3,}t соответствует фрагментам caaat, caaaat, caaaaaaaaaaat и т. д.
{n,m}	Алдыңғы элементтің n реттен m ретке дейін қайталануы	Выражение ca{2,4}t соответствует фрагментам caat, caaat и caaaat

Келтірілген мысалдардан «қайталағыштар» метасимволдары мәтіннің сәйкес орнында метасимвол алдында орналасқан символдың мүмкін болатын қайталануының санын анықтайтынын көреміз.

3.5 Анықтаушы метасимволдар

Осы топтың метасимволдары мәтін жолындағы тұрақты өрнекпен сәйкес келуді іздейтін орынды анықтайды. Жиі пайдаланылатын анықтаушы метасимволдар 3.2 кестесінде келтірілген.

3.2-кесте

Анықтаушы метасимволдар

Метасимвол	Сипаттама
^	Тұрақты өрнекпен сәйкес келетін фрагментті тек жол басынан іздеу керек
$	Тұрақты өрнекпен сәйкес келетін фрагментті тек жол соңынан іздеу керек
\A	Тұрақты өрнекпен сәйкес келетін фрагментті тек көп жолды жолдың басынан іздеу керек
\Z	Тұрақты өрнекпен сәйкес келетін фрагментті тек көп жолды жолдың соңынан іздеу керек
\b	Тұрақты өрнекпен сәйкес келетін фрагмент сөз шегінде басталады немесе аяқталады (яғни, \w және \W метасимволдарына сәйкес келетін сөздер арасында)
\B	Тұрақты өрнекпен сәйкес келетін фрагмент сөз шегінде кездеспеуі керек

Мысалы, ^cat өрнегі жол басында кездесетін cat символдарына, cat$ өрнегі — жол соңындағы cat символдарына сәйкес келеді (яғни оның соңында жолды аудару символы орналасқан), ал ^$ өрнегі бос жол, яғни бұл оның артында лезде жол соңы болатын жол басы болып табылады.

3.6 Ауыстырғыштар немесе «Символдар кластары»

Осы топтың метасимволдары мәтіндегі символдардың нақты тізбегі үшін ауыстырғыштар сияқты әрекет етеді. Жиі қолданылатын ауыстырғыштар 3.3 кестесінде келтірілген [б.356].

3.3-кесте.

Ауыстырғыштар немесе «Символдар кластары»

Символдар класы	Сипаттама	Мысал
.	Кез келген символ, \n басқа	Выражение c.t соответствует фрагментам cat, cut, c1t, c{t и т. д.
[]	Жақша ішінде жазылған тізбекке кіретін кез келген бірлік символ. Символдар диапазонын қолдануға рұқсат етілген	Выражение c[au1]t соответствует фрагментам cat, cut и c1t, а выражение c[a-z]t — фрагментам cat, cbt, cct, cdt, ..., czt
[^]	Жақша ішінде жазылған тізбекке кірмейтін кез келген бірлік символ. Символдар диапазонын қолдануға рұқсат етілген	Выражение c[^au1]t соответствует фрагментам cbt, c2t, cXt и т. д., а выражение c[^a-zA-Z]t — фрагментам cит, c1t, cЧt, c3t и т. д.
\w	Кез келген алфавиттік-цифрлік символ, яғни бас және кіші әріптер және ондық цифрлардың жиынтығынан қандайда бір символ	Выражение c\wt соответствует фрагментам cat, cut, c1t, cЮt и т. д., но не соответствует фрагментам c{t, c;t и т. д.
\W	Кез келген алфавиттік-	Выражение c\Wt соответствует фрагментам

	цифрлік емес символ, яғни бас және кіші әріптер және ондық цифрлардың жиынтығына кірмейтін қандайда бір символ	c{t, c;t, c t и т. д., но не соответствует фрагментам cat, cut, c1t, cЮt и т. д.
\s	Кез келген пробелдік символ, мысалы, пробел символы, табуляция (\t, \v), жол ауыстыру (\n, \r), жаңа бет (\f)	Выражение \s\w\w\w\s соответствует любому слову из трех букв, окруженному пробельными символами
\S	Кез келген пробелдік емес символ, яғни пробелдік жиынтыққа кірмейтін символ	Выражение \s\S\S\s соответствует любым двум непробельным символам, окруженным пробельными
\d	Кез келген ондық сан	Выражение c\dt соответствует фрагментам c1t, c2t, …, c9t
\D	Кез келген ондық сан емес символ	Выражение c\Dt не соответствует фрагментам c1t, c2t, …, c9t

3.7 Арнайы (басқарушы) символдар

Осы символдарды біз бірінші семестрде C# тілінде бағдарламалау негіздерін зерделегенде қарстырғанбыз, бірақ әдістемелік тұрғыдан оларды осы дәрісте қайталған дұрыс. Негізінен осы символдар компьютердің сырқы құрылғысына ақпаратты шығаруды басқару үшін пайдаланылады, мысалы, мониторға. Ең көп пайдаланылатын арнайы символдар 3.4 кестесінде келтірілген.

3.4-кесте.
Тұрақты өрнектердің кейбір арнайы символдары

\a	Дыбыстық сигнал (ескерту)
\b	Бір позицияға қайтару символы
\t	Көлденең табуляция
\r	Каретканы қайтару (каретка- жылжып тұратын бөлік)
\v	Тік табуляция
\f	Бетті ауыстыру
\n	Жаңа жол символы (жолды ауыстыру)
\e	Escape-символы

Ерекше жағдай: тұрақты өрнегі ішінде \b тізбегі [ ] шаршы жақшаларына алынған жиынтықтан басқа сөз шегін білдіреді, ол *забой* символын білдірерді.

3.8 Тұрақты өрнектер

Тұрақты өрнектер тілінің көмегімен жазылған символдар тізбегін тұрақты өрнектер дейміз. Әдетте, бұл қандай да бір мәтінде фрагменттерді іздестіру үшін пайдаланылатын қандай да бір үлгі. Мысалы, келесі қарапайым тұрақты өрнек бүтін сандарды сипаттау үшін арналған:

[–+]?\d+

Мұнда, «[–+]» нұсқауы осы жерде «жақшалар ішінде тізбекке жазылған кез келген бірлік символ» орналаса алатынын білдіреді, ал «?» нұсқауы – осы символ «бір рет емес немесе бір рет ғана» кездесе алатынын білдіреді. «\d» нұсқауы осы жерде «кез келген ондық цифр»

болуы мүмкін екендігін білдірсе, ал «+» нұсқауы, «алдыңғы элементтің бір немесе одан да көп қайталануы», яғни цифрдің қайталануы мүмкін екендігін білдіреді.

Нақты сандарды сипаттау үшін арналған тұрақты өрнек келесі түрге ие:

[–+]?\d+\.?\d*

Мұнда нақты санда «.» символы бар болуы мүмкін екендігі және бөлшекті бөліктің цифры «алдыңғы элементі нөл немесе одан да көп рет қайталануы» бар болуы мүмкін екендігі көрсетілген нұсқауы қосылған.

Тұрақты өрнектермен жұмыс істеуге арналған барлық типтер System.Text.RegularExpressions атаулар кеңестігінде орналасқан.

Regex класы осы атаулар кеңестігінің басты класы, оның данасы тұрақты өрнекті көрсетеді. Regex класы String класы сияқты өзгермейтін болып табылады, яғни оның данасын құрған соң түзету рұқсат етілмейді.

Regex класының негізгі қасиеттері Index және Length, оларда айқындалған сәйкес келудің орнының бастапқы мәні және ұзындығы бар болады. Табылған мән Value қасиетіне орналастырылады.

IsMatch, Match және Matches әдістері мәтінде іздестіруді атқаратын негізгі әдістер.

Regex класының IsMatch әдісі мәтінде іздестіру жүргізеді және берілген мәтінде тұрақты өрнекке сәйкес келетін фрагмент табылса, true қайтарады, ал сәйкес келетін фрагмент табылмаса false қайтарады.

Regex класының Match әдісі, IsMatc әдісіне қарағанда, берілген тұрақты өрнекпен сәйкес келген мәтіннің бірінші фрагментінің Match класының объектісін қосымша құрады.

Regex класының Matches әдісі берілген тұрақты өрнекпен сәйкес келген мәтіннің барлық фрагменттерінің топтамасы (массив) MatchCollection / класының объектісін қайтарады.

Қандай да бір мәтінде іздестірудегі осы әдістердің жұмысын қарастырайық:

```
using System;
using System.Text.RegularExpressions;
using System.Text;

namespace ConsoleApplication1
{
 class Program
 {
  static void Main()
  {
   string text    = "Язык регулярных выражение определяет
шаблоны символов.";
   if (Regex.IsMatch(text, ”ред|выр"))
Console.WriteLine("Есть совпадение!");    else
Console.WriteLine("Совпадений нет.");
   Console.WriteLine();
   Match Obj = Regex.Match(text, @”ред|выр");
   Console.WriteLine( Obj.Success);
   Console.WriteLine( Obj.Index);
   Console.WriteLine( Obj.Length);
   Console.WriteLine( Obj.Value);
   Console.WriteLine( Obj.ToString());
   //Console.WriteLine(Obj);
   Console.WriteLine();
   MatchCollection Kol = reg.Matches(text);
   Console.WriteLine("Число совпадений = {0}", Kol.Count);
```

```
        foreach(Match T in Regex.Matches(text, "ред|выр"))
         Console.WriteLine("{0} - {1}",T.Index, T.Value);
        Console.ReadLine();
       }
      }
    }
```

Бағдарлама жұмысы:

```
Есть совпадение!

True
16
3
выр
выр

Число совпадений = 2
16 – выр
28 – ред
```

Бағдарламаға деген кейбір түсініктемелер.

Regex класының IsMatch әдісін пайдаланғанда, біз қолдануға тек іздестірудің логикалық нәтижесін true немесе false аламыз.

Regex класының Match әдісін пайдалану кезінде біз өрістерінің мәнін монитор экранына шығаратын объект құрастырылады. Жолдық түрге түрлендірілген бүкіл объектінің мәні жеке жол ретінде шығарылған. Осы әдіс үшін соңғы шығару жолы өзгертілді:

```
Console.WriteLine(Obj);
```

Қателер жөнінде хабарламалар жоқ және еш өзгеріс жоқ.

Осы әдісте тұрақты өрнек алдында «@» символы орнатылған. Бұл «\» бэкслеш символынан басталатын басқарушы символдарда пайдалану кезінде C# тілінде әрекет ететін «экрандау» механизмін айналып өту үшін істелінген. Біздің тұрақты өрнекте «\» символдары жоқ, сондықтанда «@» символын қоймауға болады. Тұрақты өрнектің мәтінінде «@»префиксінсіз және «\» символының болуынсыз әр «\» символын кері қисық сызық деген төрт символмен «экрандау» қажет болушы еді. Мысалы, бізге бэкслешы бар тұрақты өрнекті - \sector жазу қажет. Бірақ, «\s» комбинациясы ауыстырғыш болып табылады, сондықтанда «\» символын экрандау қажет және де жолда «\\sector» қосарланған символ пайда болады. Бірақ, тұрақты өрнекте екі бэкслеште қайтадан экрандалуға тиіс, яғни "\\\\sector". Бір сөзбен айтқанда, бэкслешті салыстыру үшін тұрақты өрнек жолы ретінде '\\\\' жазу керек, өйткені тұрақты өрнек \\ болуға тиіс, және әр қисық сызық \\ сияқты қарапайым жолға ауыстырылуға тиіс.

Тұрақты өрнектермен жұмыс істегенде, «@» символын қолдану ұсынылады – яғни тұрақты өрнектерді @-константалар түрінде көрсету.

Regex класының Matches әдісін пайдаланғанда, қасиеттерін foreach циклының көмегімен қарастыруға болатын Match типті объектілердің массиві құрастырылады.

Біздің мысалда тұрақты өрнектер Regex класының әдістерінде талданатын мәтіннен кейін екінші параметр ретінде берілген болатын. Бірақ, Regex класының объектілерін құру (тұрақты өрнекті құру) және осы объект үшін аталған әдістерді пайдалану нұсқасы бар. Бұл жағдайда әдістерде тек бір формалды параметр бар – талданатын мәтін. Мысалы:

```
Regex reg = new Regex(@"ред|выр");
Match Obj = reg.Match(text);
```

Regex класында тұрақты өрнекті сипаттау үшін бірнеше артық жүктелген конструктор анықталған:

Regex() – бос тұрақты өрнекті құрайды;

Regex(String) – берілген тұрақты өрнекті құрайды;

Regex(String, RegexOptions) – берілген тұрақты өрнекті құрайды және RegexOptions (опциялар) тізбелеудің элементтерінің көмегімен оны өңдеу үшін параметрлер береді.

Regex класында мәтінде іздестіру әдістерімен қатар, мәтінді өңдеудің басқа да көптеген әдістері бар екенін атап кету қажет. Мысалы, Regex.Replace әдісі берілген тұрақты өрнекке сәйкес мәтін фрагменттерін ауыстырады, ал Regex.Split әдісі тұрақты өрнектерге мәтінді бөлушіні береді.

Regex класының әр әдісі анықтаушы опцияларды қолдана алады, олардың кейбіреулері 3.5 кестесінде келтірілген.

3.5-кесте.

Regex класының әдістерінің кейбір опциялары

Опция	сипаттама
CultureInvariant	Предписывает игнорировать национальные установки (культуру) строки.
ExplicitCapture	Модифицирует способ поиска соответствия, обеспечивая только буквальное соответствие.
IgnoreCase	Игнорирует регистр символов во входной строке.
IgnorePattern-Whitespace	Удаляет из строки не защищенные управляющими символами пробелы и разрешает комментарии, начинающиеся со знака фунта или хеша.
Multiline	Изменяет значение символов ^ и $ так, что они применяются к началу и концу каждой строки, а не только к началу и концу всего входного текста.
RightToLeft	Предписывает читать входную строку справа налево вместо направления по умолчанию – слева на право.

3.9 Тұрақты өрнектің элементтерін топтастыру

Тұрақты өрнектің элементтерін топтастыру үшін дөңгелек жақшалар қолданылады. Топтастыру дөңгелек жақшалардағы тұрақты өрнектің мағыналық бөлігімен сәйкес келген талданатын мәтіннің мәндерін (фрагменттерін) арнайы массивта (коллекцияда) есте сақтау үшін қолданылады.

```
        . . .
        string text = " 540-356 54-03-56 ili 49-65-78.";
        . . .
        foreach (Match T in Regex.Matches(text, @"(\d\d)-(\d\d)-
(\d\d)"))
         {
           Console.WriteLine(T);
           Console.WriteLine(T.Groups[0]);
           Console.WriteLine(T.Groups[1]);
           Console.WriteLine(T.Groups[2]);
           Console.WriteLine(T.Groups[3]);
           Console.WriteLine(T.Groups[4]);
           Console.WriteLine(T.Groups[5]);
```

```
  }
. . .
```

Бағдарлама жұмысы:

```
54-03-56
54-03-56
54
03
56


49-65-78
49-65-78
49
65
78
```

Group класы Capture класының мұрагері, сонымен бірге Match класының аталығы болады. Өзінің аталығынан Index, Length және Value қаисеттерін мұраға алады және оларды өз ол ұрпақтарына береді.

«Сәйкес келуді іздестіру барысында топтар қай кезде және қалай құрастырылатынын неғұрлым толығырақ қарастырайық. Егерде тұрақты өрнекте пайдаланылатын символдарды сипаттайтын алдыңғы кестені, жекелей алғанда топтастыру символдарына назар салып талдасақ, онда топтармен байланысты бірнеше маңызды деректі анықтаймыз:

Іздестіру шартын қанағаттандыратын бір бағыныңқы жол табылғанда, бір топ емес, топтар коллекциясы құрастырылады;

индексі 0 топта табылған сәйкес келу туралы ақпарат бар;

коллекциядағы топтар саны тұрақты өрнектің жазбасындағы дөңгелек жақшалардың санына байланысты. Дөңгелек жақшалардың әр жұбы бағыныңқы жолдың дөңгелек жақшаларда берілген үлгіге сәйкес келетін бөлігін сипаттайтын қосымша топты құрайды;

топтар индекстелген болуы мүмкін, бірақ атауланған да болуы мүмкін, өйткені дөңгелек жақшаларда топтың атауын көрсетуге рұқсат етілген.

Қорытындысында, әртүрлі ақпараты бар жолдарды талдау кезінде атауланған топтарды құру өте пайдалы екенін атап кетемін.» [Основы программирования на C# Биллиг Владимир Арнольдович б.143]

Іздестіру қорытындыларының басылымынан топтардың коллекциясы екі элементтен тұратынын көреміз. Коллекция элементіндегі топтар саны тұрақты өрнектің жазбасындағы дөңгелек жақшалар жұптарының санына сәйкес келеді (индексі 0 топта табылған элемент туралы барлық ақпарат бар). Коллекция элементінің қалған топтарында топтардың дөңгелек жақшаларында берілген үлгілерге сәйкес келетін ақпарат бар.

Бағдарлама кодында индекстері 4 және 5-ке тең «қолданыста жоқ» топтардан ақпаратты шығару қосымша жазылған. Монитор экранында бос жолдар пайда болды, бірақ қателер туралы хабарламалар жоқ.

Топтастыру жеке символ үшін немесе тұрақты өрнектегі символдардың тізбегі үшін қайталағышты беру қажет болғанда пайдаланылады.

Айнымалының (қайталағыштың) атауы дөңгелек жақшаларда немесе апострофтарда беріледі, мысалы:

(?<айнымалының атауы > тұрақты өрнектің мағыналық бөлігі)

Мысалы, мәтіннен nn–nn–nn түрінде жазылған телефон нөмірлерін анықтау қажет болсын. Нөмірді іздестіру үшін тұрақты өрнекті келесідей тәсілмен жазуға болады:

```
@"(?<num>\d{2}-\d{2}-\d{2})"
```

Осы өрнекте бұрышты жақшаларда «?» символынан соң топ атауы беріледі, содан соң топтың тұрақты өрнегінің үлгісі беріледі – \d{2}-\d{2}-\d{2}. Қандай да бір мәтінде тұрақты өрнектің үлгісі бойынша телефон нөмірлерін іздестіруді атқаратын бағдарламаның фрагменті келесі түрге ие:

```
        string text = " 540-356 54-03-56 ili 49-65-78.";
        . . .
        foreach (Match T in Regex.Matches(text, @"(?<num>\d{2}-
\d{2}-\d{2})"))
        Console.WriteLine( T.Groups["num"]);
        . . .
```

Бағдарлама жұмысы:

54-03-56
49-65-78

Мәтінді талдау кезінде табылған телефондардың нөмірлері Groups коллекциясына тізбектеліп жазылынатын болады. num (нөмір) айнымалысы алдын ала сипатталғанген емес және коллекцияның бөлігі – коллекция бірінші тобының атауы болып табылады. Коллекцияның әр тобының нөлдік индексін беріп, іздестіру нәтижелерін қарастыруға болады. Мысалы:

```
        foreach (Match T in Regex.Matches(text, @"(?<num>\d{2}-
\d{2}-\d{2})"))
         Console.WriteLine(T.Groups[0]);
```

Бағдарлама жұмысы:

54-03-56
49-65-78

Кері сілтемелерді қалыптастыру үшін топтастыруды пайдаланудың тағы бір нұсқасын қарастырамыз. Дөңгелек жақшаларға алынған барлық конструкциялар 1-ден бастап автоматты түрде нөмірленеді (нөлдік индекс бүкіл тұрақты өрнекті белгілеу үшіне қолданылады). Алдын ала кері қисық сызықпен белгіленген осы нөмірлерді сәйкес конструкцияға сілтеме үшін пайдалануға болады. Мысалы, (\w)\1 өрнегі, нақты айтқанда @”(\w)\1”, бірдей әліпби-цифрлық символдардың барлық жұптарын табуға мүкіндік береді. Әдетте осындай өрнек сөздерде қосарланған символдарды іздестіру үшін пайдаланылады, мысалы, class, mass.

```
. . .
        string text = "Язык регуляррных выражжений опредееляет
шшаблоны символов.";
                . . .
        foreach (Match T in Regex.Matches(text,@"(\w)\1"))
          Console.WriteLine("{0} - {1}", T.Index, T.Value);
                . . .
```

Бағдарлама жұмысы:

12 - рр
23 - жж
35 - ее
42 - шш

Атауы бұрышты жақшалардағы өрнекте берілетін айнымалыны сондай-ақ өрнектің кейінгі бөлігіне кері сілтемелер үшін қолдануға болады. Мысалы, сөздердегі қосарлы символдарды іздестіруді @”((?<s>\w)\k<s>)” өрнегінің көмегімен атқаруға болады. Осы тұрақты өрнекі қарастырайық. Топ «(?<s>\w)» өрнегімен берілген. Бұрышты жақшалардағы сұрақ белгісінен кейін «<s>» топтың атауы берілген. Топ тауынан кейін осы топты сипаттайтын үлгі орналасады, біздің мысалда үлгі «\w» өрнегімен беріледі. Осы өрнек топ бір кез келген әліпби-цифрлық символдан, яғни кіші және үлкен әріптердің және ондық цифрлардың жиынтығындағы символдан тұратынын білдіреді. Топтың сипатталуынан соң кері сілтеменің белгісі «\k» символдарының жұбы орналасқан. Осы жұптан соң «<s>» деген топ атауы орналасқан.

```
. . .
        string text = "Язык регулярррных выражжений определяет
шаблоны символов.";
                . . .
        foreach (Match T in Regex.Matches(text,
@"((?<s>\w)\k<s>)"))
         {
           Console.WriteLine(T.Groups["s"]);
           Console.WriteLine(T.Groups[0]);
         }
                . . .
```

Бағдарлама жұмысы:

```
р
рр
ж
жж
е
ее
ш
шш
```

«Топ атауы» үшін және нөлдік индексті топ үшін іздестіру нәтижелерін шығару әртүрлі. Топ атауына тек бір символ – үлгі сәйкес келсе, нөлдік индексті топқа табылғанның барлығы сәйкес келеді.

Символ регистрін ескермей іздестіруді ұйымдастыруға болады:

```
. . .
        string text = "Язык регулярррных выражжений опредееляет
шшаблоны символов.";
                . . .
        foreach (Match T in Regex.Matches(text,
                                @"((?<s>\w)\k<s>)",
                        RegexOptions.IgnoreCase)
         {
           Console.WriteLine(T.Groups["s"]);
           Console.WriteLine(T.Groups[0]);
         }
                . . .
```

Бағдарлама жұмысы:

```
Я
Яя
р
рр
```

ж
жж
е
ее
ш
шш

Мәтінде қатарынан орналасқан және пробелдардың ерікті санымен бөлінген қайталана беретін сөздерді іздестіру үшін (регистріне қарамастан) тұрақты өрнектің келесі конструкторын пайдалануға болады:

```
. . .
        Regex reg = new Regex(@"\b(?<word>\w+)\s+(\k<word>)\b",
RegexOptions.IgnoreCase);
        string text = "Class     class Regex это Это class regex
регулярных выражений.";
                . . .
        foreach (Match T in reg.Matches(text))
         {
           Console.WriteLine(T.Groups["word"]);
           Console.WriteLine(T.Groups[0]);
         }
                . . .
```

Бағдарлама жұмысы:
Class
Class class
это
это Это

Мәтінде жолдың кез келген позициясында орналасқан қайталана беретін сөздерді (регистрге қарамастан) іздестіру үшін, « бөлгіш пробелдарды– \s» кез келген символдарға – «.» ауыстыру керек.

```
. . .
        Regex reg = new Regex(@"\b(?<word>\w+).+(\k<word>)\b",
RegexOptions.IgnoreCase);
        string text = "Class     class Regex это Это class regex
регулярных выражений.";
                . . .
        foreach (Match T in reg.Matches(text))
         {
           Console.WriteLine(T.Groups["word"]);
           Console.WriteLine(T.Groups[0]);
         }
                . . .
```

Бағдарлама жұмысы:
Class
Class class Regex это Это class

Іздестіру нәтижесі – олар көп болу керек сияқты болса да, ол тек бір топ. «.+» қайталағышының (оның жиі жағдайда тойымсыз деп атайды «greedy») жұмысына назар аударайық, ол алдыңғы элементтің бір немесе одан да көп қайталануын білдіреді. Мәтіннің бірінші сөзі үшін, ал ол «Class» сөзі, қайталағыш мәтіннің соңына дейін барлық сәйкес

келулерді іздестіреді де және табылған фрагмент мәтіннен «кесілініп алынады» да кейінгі талдауға қатыспайды.
Мысалымыздың мәтінін келесі түрде өзгертейік:

```
        string text = "Class Это class Regex это Это  regex это
регулярные Это  выражения.";
```

Бағдарлама жұмысы:
Class
Class Это class
Regex
Regex это Это regex
это
это регулярные Это

Іздестіру нәтижесінде коллекцияның үш тобы табылды.

3.10 Тұрақты өрнектерде Split және Replace әдістерін пайдалану

Regex класында мәтінде іздестіру әдістерімен қатар, мәтінді өңдеудің кейбір басқа әдістері де бар. Regex класының іздестіру жүргізбейтін әдістерінің арасында Split әдісі жиі қолданылады. Ол тұрақты өрнектерге мәтінді бөлшектегішті беруге мүмкіндік береді. Екінші жиі қолданылатын әдіс Replace, ол берілген тұрақты өрнекке сәйкес мәтін фрагменттерінің ауыстыруын атқарады.

Оқу мысалдарында осы әдістердің пайдаланылуын қарастырамыз.

Осы тіліндегі қандай да бір мәтіні кодтау үшін 00-ден 32-ге дейінгі (регистрдің мәні есепке алынбайды) ондық сандар қолданылады деп болжаймыз. «Үтір» символы 33 санымен, «нүкте» –34 санымен, ал «пробел» символы 35 санымен кодталады. Мәтіннің қалған символдары, яғни цифрлармен басқа әліпбилердің әріптері 40-тан 99-ға дейінгі диапазонда кездейсоқ сандармен кодталады. Кодталған мәтінде сандар «пробел» символымен бөлінеді. Алғашқы мәтінді келтірілген ережелердің көмегімен кодтап, оны жаңа уақытша мәтінге жазу қажет. Уақытша мәтіннің декодтауын қарастыру қажет. Декодтау 00-ден 35-ке дейінгі сандарды түрлендіруді келтірілген ережелерге сәйкес орындау керек. 40-тан 99-ға дейінгі диапазондағы сандарды елемеу керек. Монитор экранында алғашқы, кодталған және декодталған мәтіндерді көруді қарастыру қажет. Бағдарламада Regex класының Split және Replace әдісін пайдалану керек.

«PreobText» деген жеке клас құрамыз, онда конструкторды және мәтінді кодтау және декодтаудың жолдық әдістерін ұйымдастырамыз. Мәтіннің өзі конструкторға клас өрістерінің инициализациялануы кезінде берілетін болады.

Мәтінді кодтау әдісіне Regex класының Split әдісінің көмегімен алғашқы мәтінді «сөздерге» «бөлу» кіреді. Әрі қарай тапсырмаға сәйкес мәтіннің әр «сөзінің» кодталауы орындалады. Алынған екі разрядты сандардың тізбектері жаңа мәтінге «біріктіріледі», ол әдіспен бағдарламаға қайтарылады.

Декодтау әдісі екі разрядты сандардың тізбегін Regex класының Replace әдісін пайдалана отырып жаңа мәтінге түрлендіреді. Мәтін әрі қарай тапсырмаға сәйкес декодталады.

Бағдарлама жұмысының әр кезеңінде (бағдарламаның менюі) меню пункті жұмысының нәтижесін басу қарастырылған.

Бағдарлама коды:

```
using System;
using System.Collections.Generic;
```

```
using System.Text.RegularExpressions;
using System.Text;

namespace ConsoleApplication1
{
 class Program
 {
  class PreobText
  {
   string text;
   public string rez;
   string alfavit = "абвгдеёжэийклмнопрстуфхцчшщъыьэюя,.
";

   //Конструктор
   public PreobText(string txt)
   {
    text = txt;
    rez = "";
   }
   // метод кодирования
   public string Codirovanie()
   {
    bool ok = false;
    text = text.ToLower();
    int m = 0, k = 0, Kol = 0, r = 0;
    Random rnd = new Random();
    string p, clo;
    //Console.WriteLine("nacalo preobrazovanij!");
    //Console.WriteLine(text);
    //Console.WriteLine(rez);
  List<string> clova=new List<string>(Regex.Split(text,@"[
]"));
    m = clova.Count;
    string[] newclova = new string[m];
    foreach (string ss in clova)
    {
     Kol = ss.Length;
     //  Console.WriteLine(Kol);
     clo = "";
     if (Kol != 0)
     {
      for (int j = 0; j < Kol; j++)
      {
       ok = false;
       for (int i = 0; i < 36; i++)
       {
        if (ss[j] == alfavit[i])
        {
         ok = true;
         if (i <= 9)
         { p = "0" + i.ToString() + " "; }
```

```
                    else p = i.ToString() + " ";
                   clo = clo + p;
                  }
                 }
                 if (ok == false)
                 {
                  r = rnd.Next(40, 99);
                  clo = clo + r.ToString()+ " ";
                 }
                }
                clo.Trim();
                if (clo != "") { newclova[k] = clo; k++; }
                //Console.WriteLine(newclova[k - 1]);
               }
              }

              // Console.WriteLine("konec preobrazovanij!");
              // Console.WriteLine();
              for (int j = 0; j < m; j++)
              {
               //Console.WriteLine(newclova[j]);
               rez = rez + newclova[j] + "35" + " ";
              };
              return rez;
             }

             // метод декодирования
             public string Decodirovanie()
             {
              string dec = "";
              int i=0,Kol=0;
              dec = Regex.Replace(rez, @"\d{2}", @"<$0");
              Console.WriteLine(dec);
              List<string> clova=new
List<string>(Regex.Split(dec,@"<"));
              Kol = clova.Count;
              // Console.WriteLine(Kol);
              rez = "";
              foreach (string ss in clova)
              {
               Kol = ss.Length;
               if (Kol != 0)
               {
                i = Convert.ToInt32(ss);
                if (i < 36) rez = rez + alfavit[i];
               }
              }
              // Console.WriteLine("Konec");
              return rez;
             }
            }
```

```
            static void Main(string[] args)
            {
             string txt = "CG1но34в55а! н41ас веGamesдуZ23т куTiд44а
8767 то, с62нkwовtta я tackалщу росюакомзукак.";
             PreobText ob = new PreobText(txt);
             string ntxt;
             int w = 0;
             string buf;
             while (w < 4)
             {
              Console.WriteLine("1 - Просмотр исходного текста");
              Console.WriteLine("2 - Кодирование текста");
              Console.WriteLine("3 - Декодирование текста");
              Console.WriteLine("4 - Выход из программы");
              Console.WriteLine("Введите пункт меню программы");
              buf = Console.ReadLine();
              w = Convert.ToInt32(buf);
              switch (w)
              {
               case 1: Console.WriteLine(txt); break;
               case 2:
                      {
                       ntxt = ob.Codirovanie();
                       Console.WriteLine(ntxt);
                       break;
                      }
               case 3:
                      {
                       ntxt = ob.Decodirovanie();
                       Console.WriteLine(ntxt);
                       break;
                      }
              default: break;
             }
            }
           }
          }
```

Бағдарлама жұмысы:

```
1 - Просмотр исходного текста
2 - Кодирование текста
3 - Декодирование текста
4 - Выход из программы
Введите пункт меню программы
1
CG1но34в55а! н41ас веGamesдуZ23т куTiд44а 8767 то, с62нkwовtta я tackалщу
росюак
омзукак.
1 - Просмотр исходного текста
2 - Кодирование текста
3 - Декодирование текста
4 - Выход из программы
```

Введите пункт меню программы
2
18 41 79 14 15 50 87 02 80 59 00 53 35 14 59 60 00 18 35 02 05 70 54 88 77 47 04
20 43 47 77 19 35 11 20 55 98 04 46 52 00 35 46 70 97 76 35 19 15 33 35 18 54 9
0 14 68 79 15 02 54 77 00 35 32 35 19 00 85 86 62 56 26 20 35 17 91 54 31 98 11
74 79 72 92 45 00 11 34 35
1 - Просмотр исходного текста
2 - Кодирование текста
3 - Декодирование текста
4 - Выход из программы
Введите пункт меню программы
3
<18 <41 <79 <14 <15 <50 <87 <02 <80 <59 <00 <53 <35 <14 <59 <60 <00 <18 <35
<02
<05 <70 <54 <88 <77 <47 <04 <20 <43 <47 <77 <19 <35 <11 <20 <55 <98 <04 <46
<52
<00 <35 <46 <70 <97 <76 <35 <19 <15 <33 <35 <18 <54 <90 <14 <68 <79 <15 <02
<54
<77 <00 <35 <32 <35 <19 <00 <85 <86 <62 <56 <26 <20 <35 <17 <91 <54 <31 <98
<11
<74 <79 <72 <92 <45 <00 <11 <34 <35
снова нас ведут куда то, снова я тащу рюкзак.
1 - Просмотр исходного текста
2 - Кодирование текста
3 - Декодирование текста
4 - Выход из программы
Введите пункт меню программы

4 C# ТІЛІНДЕ ҮДЕРІСТЕ ТІЗБЕКТЕР АРАСЫНДА ДЕРЕКТЕРДІ ЖІБЕРУ

4.1 Жүйелік бағдарламалаудағы тізбек түсінігі

4.1.1 Тізбек түсінігі

Компьютер процессорының жұмысы бағдарлама компиляциясынан кейін қалыптасатын командаларды (нұсқаулықтарды) орындау болып табылады. Бұл бағдарлама нұсқаулықтарының тізбектілігі бағдарламаны басқару командаларының ағыны деп аталады. «Басқару ағыны» термині C++ тілінде пайдаланылады. C# тілінде бағдарламаны басқару командаларының ағыны тізбек деп алынған - компьютер процессоры атқаратын нұсқаулықтар «тізбектелген» тізбекпен (thread) салыстырады.

Егер операциялық жүйеде бір тізбек ғана бар болса, онда операциялық жүйе бір бағдарламалық деп аталады.

Егер операциялық жүйеде бір мезетте әртүрлі бағдарламалардың бірнеше тізбектері болса, онда ол мультибағдарламалық деп аталады.

Компьютер жұмысының тиімділігі көбінесе компьютер процессорының жүктемесімен анықталады. Сондықтанда, мультибағдарламалық ОЖ маңызды міндеті процессордың тиімді жүктемесін қамтамасыз ететіндей етіп, бағдарламалар тізбектерін диспетчерлеу.

Бағдарламаның кез келген тәуелсіз орындалатын фрагменті бағдарлама тізбегі бола алады.

a және b сандарының қосындысын есептейтін бағдарлама мысалын қарастырайық.

```
using System;
namespace ConsoleApplication1
{
 class Program
 {
  static void Main(string[] args)
  {
   int a,b,c;
   string buf;
   Console.Write("Введите целое значение a ");
   buf = Console.ReadLine();
   a = Convert.ToInt32(buf);
   Console.Write("Введите целое значение b ");
   buf = Console.ReadLine();
   b = Convert.ToInt32(buf);
   c = a + b;
   Console.WriteLine("a+b={0}", c);
   Console.WriteLine("Для продолжения нажмите клавишу
Enter");
   Console.ReadLine();
  }
 }
}
```

Осы бағдарламаның әр әрекеті әрдайым бір мағыналы анықталған және бірінен кейін бірі орындалады – бағдарламада тек бір ғана тізбек бар.

a және b сандарының қосындысын есептейтін функцияны енгізу арқылы бағдарламаны өзгертеміз.

```
using System;
namespace ConsoleApplication1
{
 class Program
 {
  public static int sum(int a, int b)
  { return a + b; }
  static void Main(string[] args)
  {
   int a, b, c;
   string buf;
   Console.Write("Введите целое значение a ");
   buf = Console.ReadLine();
   a = Convert.ToInt32(buf);
   Console.Write("Введите целое значение b ");
   buf = Console.ReadLine();
   b = Convert.ToInt32(buf);
   c = sum(a,b);
   Console.WriteLine("a+b={0}", c);
   Console.WriteLine("Для продолжения нажмите клавишу
Enter");
   Console.ReadLine();
  }
 }
}
```

Егерде бағдарламаны орындау кезінде **Main** функциясы **sum** функциясының орындалуын тосса, оонда бағдарлама бір тізбекпен қалады.

Егерде бағдарламаны орындау кезінде **Main** функциясы **sum** функциясының орындалуын тоспай, орындалауын жалғастыра берсе, онда бағдарламада екі тізбек пайда болады – басты, **Main** функциясының тізбегі және **sum** функциясының тізбегі. Осы жағдайда **Main** функциясымен монитор экранына шығарылған **c** айнымалысының мәні **a және b** айнымалылардың қосындысына сәйкес келмеуі мүмкін.

Шартты өтудің операторларын пайдаланатын бағдарламаларда бұдан да көп тізбек болуы мүмкін.

Бағдарламаның жұмысын дұрыс ұйымдастыру үшін тізбектер арасында деректерді беруді, деректерді бірігіктіріп пайдалануды және әртүрлі тізбектердің немесе тізбектік функциялардың жұмысын синхрондауды (ретке келтіру) үйрену керек.

4.1.2 Параллельді тізбектермен пайдаланылатын әдістерге қойылатын талаптар

Параллельді тізбектермен «қауіпсіз» шақыруға болатын әдістер қандай талаптарды қанағаттандыруы керектігін қарастырамыз.

Бұл үшін тізбек контексті түсінігін - тізбек өзінің орындалуы кезінде өзі жүгіне алатын компьютер жадысының облысы ретінде анықтаймыз.

Келесі функцияның жұмысын қарастырамыз:

```
public static int funk1(int n)
{
 if (n >= 0) n --;
 if (n  < 0) n ++;
```

```
  return n;
}
```

Осы функцияны шақыру кезінде **n** айнымалының көшірмесі құрылады. Осы көшірме шартқа сәйкес өзгертіледі және нәтижесі тізбекке қайтарылады. Әрі қарай айнымалының көшірмесі жадыдан жойылады. Осындай функция тізбек үшін қауіпсіз деп аталады.

funk1 функциясын ол **n** ауқымды айнымалысымен жұмыс істей алатындай етіп өзгертеміз.

```
int n;
. . .
public static void funk2()
{
  if (n >= 0) n --;
  if (n  < 0) n ++;
}
```

funk2 функциясын параллельді түрде бірнеше тізбек шақыртса **n** айнымалысының мәні орынсыз өзгеруі мүмкін (**n** айнымалысы бірден көп өзгертілуі орын алуы мүмкін). Осындай функциялар тізбек үшін қауіпсіз емес деп аталады.

Бағдарламалаудың кейбір тілдерінде, мысалы, C++ тілінде, функцияның статикалық айнымалылары түсінігі бар. Статикалық айнымалылар функциялар ішінде (жергілікті айнымалылар) жарияланады, бірақ осы айнымалылардың мәні функция жұмысының соңында жадыда сақталынады және функцияның қайта шақыртуда пайдаланылуы мүмкін. Яғни осындай статикалық айнымалы функция ішінде бағдарламаның ауқымды айнымалысы сияқты әрекет етеді. Бұл сияқты функцияға параллельді түрде бірнеше тізбек қатынаса, онда статикалық айнымалының мәні қандай да бір тізбектің функцияларын шақырту санына сәйкес келмеуі мүмкін.

Жалпы жағдайда функция келесі талаптарды қанағаттандырса, қауіпсіз немесе реентерабелді деп аталады:

- мәндері параллельді жұмыс істеп тұрған тізбектермен өзгертілетін ауқымды айнымалыларды пайдаланбаса;
- функция ішінде анықталған және қатарлас жұмыс істеп тұрған тізбектермен өзгертілетін статикалық айнымалыларды пайдаланбаса;
- функция ішінде анықталған статикалық деректерге нұсқағыштарды қайтармаса.

Егерде, тізбектер бірігіп қолданатын, әртүрлі тізбектер функцияларының ресурстарына блоктаудың әртүрлі тәсілдерін пайдаланса, қауіпсіз функцияларға қойылатын аталған талаптарды қамтамасыз етуге болады.

4.1.3 Тізбектер күйі

Тізбектер күйі компьютер процессорының тізбектерден тұратын бағдарламамен жұмыс істеуге дайындығына және бағдарламаның өз дайындығына тәуелді (байланысты) болады.

Процессордың дайындығы оның осы бағдарламаны орындауға «бөлуімен» (айырықшалауымен) анықталады – яғни процессор осы бағдарламамен жұмыс үшін уақыт квантын алады.

Бағдарламаның дайындығы оның бағдарламаны орындау үшін қажетті барлық ресурстарды алуымен байланысты. Процессор мен бағдарламаға байланысты ағынның келесі күйлері мүмкін болуы мүмкін:

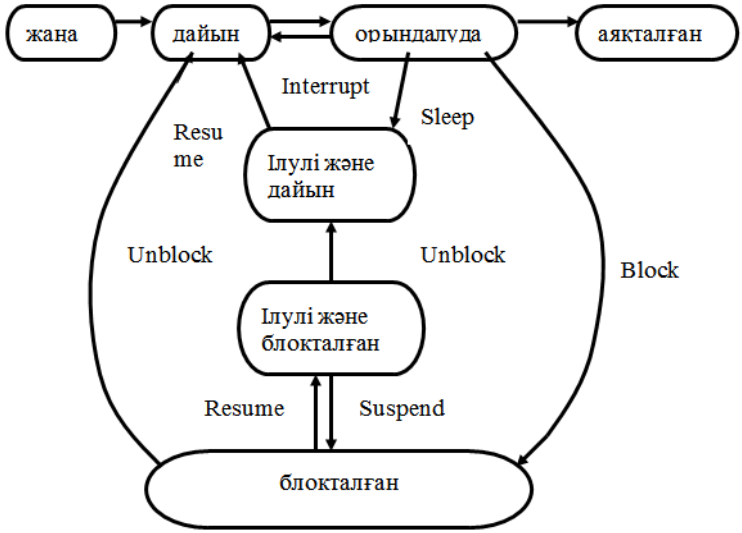
- тізбек орындауға дайын («бөлінбеген» (жеке алынған), «дайын»);
- тізбек блокталған («бөлінбеген», «дайын емес»);

- тізбек орындалуда («бөлінген», «дайын»).

Әдетте, тізбектің аталған күйлеріне тағы екі күй қосылады – жаңа (тізбек құрылуда, бірақ дайын емес) және аяқталған (тізбек жұмысының аяқталуына сәйкес келетін күй).

Жұмысы өзара байланысты болатын бірнеше тізбектен тұратын бағдарламаларда тағы да екі тізбек болады –тізбектің орындалуын тоқтату және тізбектің орындалуын жаңғырту .

Тізбектің барлық аталған күйлерін ескере отырып, оның реттілігінің ***(поведения)*** келесі моделін салуға болады.



Сурет- 4.1 –- Тізбектің жеті күйінің моделі

Create операциясы тізбекті «жаңа» күйінен «дайын» күйіне ауыстырады.

Run операциясы тізбекті «дайын» күйінен «орындалуда» күйіне ауыстырады - тізбекке процессорлық уақыт бөлінеді.

Exit операциясы тізбекті «орындалуда» деген күйінен «аяқталды» күйіне ауыстырады.

Interrupt операциясы тізбекті «орындалуда» деген күйінен «дайын» күйіне ауыстырады - әдетте осы операция тізбектің орындалуына бөлінген процессорлық уақыт аяқталғанда тізбек үстінен атқарылады.

Resume операциясы тізбектің орындалуын қайта қалыпқа келтіреді.

Sleep операциясы тізбектің орындалуын тоқтатып қояды.

Unblock операциясы тізбекті «блокталған» күйінен «дайын» күйіне ауыстырады.

Block операциясы тізбекті «орындалуда» күйінен «блокталған» күйіне ауыстырады – әдетте, осы операция тізбек қандай да бір оқиғаның болуын тосып тұрған кезде, тізбек үстінен орындалады, мысалы, деректердің енгізілуін немесе қандай да бір ресурстың босауын тоқсан кезде.

Suspend операциясы тізбектің орындалуын тоқтатып қояды.

4.1.4 Тізбектердің диспетчерленуі және жоспарлануы

Тізбектің диспетчерленумен жоспарлану қағидаларын түсіндіру үшін компьютерде тек бір процессор бар деп санаймыз.

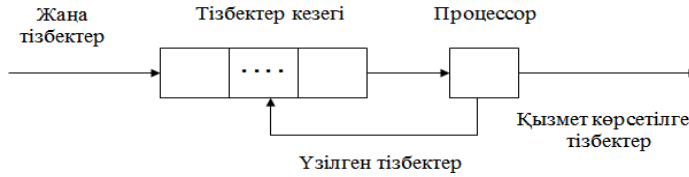
Процессор жұмысының мультибағдарламалық режімін ұйымдастыру үшін оның жұмыс уақыты кванттарға – тізбектерге олардың жұмысы үшін бөлінетін аралықтарға бөлінеді (Windows-та ол шамамен 20 – 30 миллисекунд). Уақыт кванты аяқталған соң тізбектің орындалуы үзіледі де, процессор басқа тізбекке тағайындалады. Тізбектер арасында уақыт кванттарын үлестірумен арнайы бағдарлама - тізбектер менеджері шұғылданады.

Тізбектер менеджері процессорды басқа тізбектің орындалуына ауыстырғанда, ол келесі әрекеттерді орындауға тиіс:

- үзілетін тізбектің контекстін сақтауға;
- іске қосылатын тізбектің контекстін қалпына келтіруге;
- іске қосылатын тізбектің басқаруын тапсыруға.

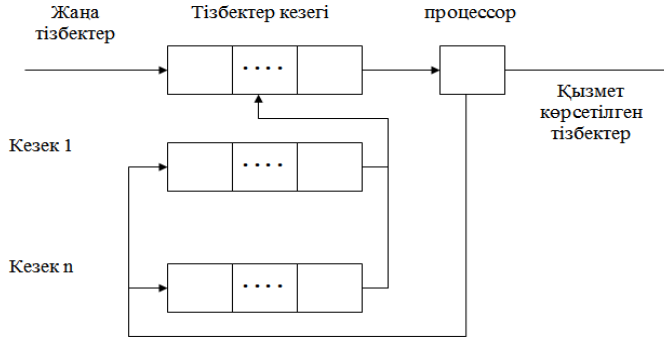
Егерде барлық қызмет көрсетілетін тізбектердің орындау басымдылықтары бірдей болса, онда тізбектерді басқару алгоритмі FIFO (first in – first out) бірінші келіп – бірінші шығасын қағидаты бойынша «кезектің» көмегімен жүзеге асырылады, және үзілген тізбектер кезек соңына тұрады.

Тізбектерге қызмет көрсету сызбалық түрде сурет 4.2 берілген.



Сурет 4.2 – Тізбектерге қызмет көрсету

Егерде тізбектер әртүрлі басылымдықтарға ие болса, онда басымдылықтары бірдей тізбектер кезектері қалыптастырылады. Бірнеше кезекке қызмет көрсетудің ең қарапайым алгоритмі олардың тізбектерінің басымдылықтарын ескере отырып кезектерге қызмет көрсетуде негізделген. Тізбектерге қызмет көрсету сызбалық түрде 7.3 суретінде көрсетілген.



Сурет 4.3 – Тізбектерінің басымдылықтары әртүрлі бірнеше кезекке қызмет көрсету

Тізбектерді басқару алгоритмдері жүйенің келесі параметрлерін ескеруге тиіс:
- микропроцессорды жүктеу уақыты максималды болуға тиіс;
- тізбектің жүйеде болу уақыты минималды болуға тиіс;
- тізбектердің күту уақыты минималды болуға тиіс;
- жүйенің өткізгіштік қабілеттілігі максималды болуға тиіс;
- жүйенің өтінімге қызмет көрсету реакциясының уақыты минималды болуға тиіс.

Көп процессорлы компьютерлерде тізбектер менеджері әртүрлі тізбектер кодты әртүрлі процессорларда орындаған кезде тізбектер жұмысының уақытын кванттеуді де, процессорлар жұмысының параллельдігін де пайдаланады.

Уақытты кванттеу қажеттілігі бәрібір қалады, өйткені операциялық жүйе өзінің жеке (жүйелік) тізбектерімен қатар, басқа қосымшалардың тізбектеріне де қызмет көрсетуге тиіс.

4.1.5 Тізбектерді анықтау

Windows-та ағын/тізбек дегеніміз – ядро объектісі, оған қосымшаны орындау үшін ОЖ процессорлық уақытты бөледі.

Windows-ғы қосымша бір немесе бірнеше үдерістен тұруы мүмкін. Әр үдеріс бір немесе бірнеше тізбектер тудыруы мүмкін. Әр тізбекке операциялық жүйенің кейбір ресурстары бөлінеді, олар объект түрінде ұсынылады.

Windows-та тізбек дегеніміз операциялық жүйе жүйелік ресурстармен жұмыс істеу үшін процессорлық уақыт бөлетін объект.

Әр тізбекке келесі ресурс тиесілі:
- орындалатын функцияның коды;
- процессор регистрларының жиынтығы;
- қосымшаның жұмысы үшін стек;
- операциялық жүйенің жұмысы үшін стек;
- қауіпсіздік жүйесі үшін ақпараты бар қатынау маркері.

Осы ресурстардың барлығы **тізбек контекстін** құрайды.

Windows жүйесіндегі кез келген объектіге өзінің тіркеу нөмірі - дескрипторы беріледі. Windows-та әр тізбектің дескрипторынан басқа оған өз идентификаторы беріледі. Тізбектердің идентификаторлары қызметтік бағдарламаларымен пайдаланылады және тізбектердің жұмысын бақылауға мүмкіндік береді.

Windows-та тізбектердің екі түрін ажыратады - жүйелік және қолданушы тізбектері.

Жүйелік тізбектерді операциялық жүйенің ядросы іске қосады және олар әртүрлі жүйелік міндеттерді орындау үшін қызмет етеді.

Қолданушы тізбектері қосымшалармен іске қосылады және қолданушының тапсырмаларын шешу үшін қызмет етеді.

.NET технологиясында ресурстарды бөлудің (айрықшалаудың) бұл сызбасы біраз өзгертілген - .NET қосымшасының әр үдерісі қосымшаның бір немесе бірнеше домендерінен тұрады, ал домендердің шегінде бір немесе бірнеше тізбек жұмыс істеуі мүмкін (объект сияқты тізбектің барлық қасиеттері сақталынады).

Әр доменнің ресурстары бір-бірінен оқшаланған. Қосымшаның әртүрлі домендері ешқандай деректерді бірге қолдана алмайды, ол статикалық өрістер немесе тек ортақ доменнің шегінде әртүрлі тізбектермен қолданылуы мүмкін ауқымды айнымалылар болсын олар бірге қолданылмайды.

Сонымен, домен дегеніміз ол үдеріспен тізбек арасында құрастырылатын .NET-гі «оқшаулаудың» немесе деректерді «қорғаудың» қосымша деңгейі.

.NET-гі қандай да бір қосымшалар үшін домен шегінде ***тізбектердің пулдарын*** құруға болады. ***Тізбектердің пулдары*** (ортақтастық) - ағымдағы тізбектер үшін Thread объектісін құру және қосудың орнына ThreadPool.QueueUserWorkItem әдісімен құрастырылатын арнайы шағын кезектер.

Әдетте, тізбек пулының диспетчері пулдағы тізбектер санын автоматты түрде белгілейді (бастапқыда осы мән 50 тең болады). Өздігінен пулдағы тізбектер санының жоғары шегін ThreadPool.SetMaxThreads әдісімен белгілеуге болады.

Жалпы пулдар параллелді жұмыс істеп тұрған қосымшалардың ерікті санының жұмысын ұйымдастыру қажет болғанда пайдаланылады, мысалы, веб-серверлерде. Web Services, ASP.NET, WCF серверлердің және т.б. жұмысында тізбектердің пулдары автоматты түрде бөлінеді.

4.1.6 Тізбектердің құрылуы

C# тілінде кез келген бағдарлама іске қосылғанда автоматты түрде бағдарламаның басты тізбегі құрылады. Консолдық қосымшада бағдарламаның басты тізбегі Main әдісіне сәйкес келеді.

Қосымша тізбекті іске қосу үшін Thread класының объектісін құру қажет. Тізбек объектісін құрған кезде Thread класының конструкторына ThreadStart әдісі сипатталған делегат беріледі. ThreadStart әдісі делегат әдістің формальді параметрі ретінде аты көрсетілген әдісті қосымша тізбекте іске қосуға мүмкіндік беретін әдіс. Мысалы,

```
Thread Pervij_dop = new Thread(new ThreadStart(Poick));
```

мұнда Pervij_dop – бірінші қосымша тізбектің атауы;

Poick –қосымша тізбекте іске қосылатын әдістің атауы.

Жалпы алғанда, делегат «бастапқы» бойынша жұмыс істейді және тізбек объектісін құру келесідей болады:

```
Thread Pervij_dop = new Thread(Poick);.
Pervij_dop.Start();
```

Тізбектермен жұмыс істейтін консольдік қосымшаны құрып, System.Threading атаулар кеңестігін қосу қажет.

Thread класында қасиеттер мен әдістер жиынтығы бар, олардың кейбіреулері келесі тізімде көрсетілген:

CurrentThread –қазіргі уақытта орындалып жатқан тізбекке сілтемені қайтаратын, тек оқу үшін арналған статикалық қасиет;

Sleep() – бұл статикалық әдіс, ол ағымдағы тізбектің орындалуын қолданушы белгіленген уақытқа тоқтата тұрады;

IsAlive – тізбектің қосылып тұруына немесе қосылмауына байланысты **true** немесе **false** қайтаратын қарапайым қасиет (тізбек объектісінің құрылуын талап етеді);

IsBackground – тізбектің фондық болуын көрсететін мәнді алу немесе орнату үшін арналған қарапайым әдіс;

Name – тізбектің достық мәтіндік атауын орнату үшін арналған қарапайым әдіс;

Priority – тізбектің басымдылығын алуға/орнатуға мүмкіндік беретін қарапайым әдіс (ThreadPrority тізбелеулеріндегі мәндер қолданылады);

ThreadState – тізбектің күйі жөнінде ақпарат алуға мүмкіндік беретін қарапайым әдіс (ThreadState тізбелеулеріндегі мәндер қолданылады);

Interrupt() – ағымдағы тізбектің жұмысын үзетін қарапайым әдіс;

Join()– басқа тізбектің пайда болуын немесе көрсетілген уақыт аралығында тосатын қарапайым әдіс;

Resume() –тізбектің тоқтатылып тұруынан кейін жұмысын жалғастыратын қарапайым әдіс;

Start() –ThreadStart делегатымен анықталған тізбектің орындалуын бастайтын қарапайым әдіс;

Suspend() – тізбектің орындалуын тоқтатып тұратын қарапайым әдіс. Егерде тізбектің орындалуы тоқтатылып тұрған болса, сонда әдіс ескерілмейді.

Thread класының барлық қасиеттері мен әдістері осы дәрістің қосымшасында келтірілген.

Қосымша тізбек объектісінің құрылуын және оның бағдарламаның басты тізбегінен қосылуының оқу мысалын қарастырамыз. Басты және қосымша тізбектің жұмысын көзбен көріп бақылау үшін оларда дәстүрлі түрде кейбір символдарды немесе цифраларды шығарумен for циклы қолданылады.

Дәстүрлерден алыс кетпейік:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Thread1
    {
        static void WriteCimB()
        {
            for (int i = 1; i <= 100; i++)
Console.Write("B");
        }
        static void Main()
        {
            Thread t_dop = new Thread(WriteCimB);
            t_dop.Start();
            for (int i = 1; i <= 100; i++)
Console.Write("A");
            Console.ReadLine();
        }
    }
}
```

Бағдарлама жұмысы:

```
AAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

Басты тізбек консольдік терезеге 100 «А» символын шығару үшін арналған, бір мезетте ол сондай-ақ консольдік терезеге «В» деген бір символын 100 рет шығаратын

қосымша тізбекті құрады. Бағдарлама жұмысынан олар бір мезетте жұмыс істейтінін көреміз – **for** циклдары тізбектерге бөлінетін уақыт кванттарымен үзіледі. Бір процессорлы компьютерде әр тізбекке жұмыс үшін уақыт кванты белгіленеді (Windows-та ол 20 мс.).

Бағдарламаға келесі жолдарды қосып, қосымша тізбектің күйін зерттеуге болады:

```
        Console.WriteLine("состояние доп нити ={0}",
t_dop.IsAlive);
        Console.WriteLine("приоритет доп нити ={0}",
t_dop.Priority);
```

Бағдарлама жұмысы:
состояние доп нити =True
приоритет доп нити =Normal
AAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBB

4.2 Бір үдерістің әртүрлі тізбектерінің деректерді пайдалануы

4.2.1 Әртүрлі тізбектердің жергілікті айнымалыларды пайдалануы

Алдыңғы дәрісте монитор экранының консольдік терезесіне «А» және «В» символдарын шығарған екі тізбектің жұмысын қарастырдық. Тізбектер бір-біріне тәуелсіз жұмыс істеді. C# тілінде әр объектіге деректердің өз стегі бөлінген және жергілікті айнымалылар бөлек сақталғандықтан осылай болады.

Егер сандарды консольдік терезеге шығаратын әдісі бар қандай да бір клас үшін объект (жергілікті айнымалы құрса), онда әртүрлі тізбектердің жұмысы барысында осы кластың объектісінің жергілікті айнымалысын өзгертуге бола ма, жоқ па екендігін тексеруге мүкіндік бар. Мұны біздің объектінің әдісін басты тізбекте және бір қосымша тізбекте іске қосып, ұйымдастаруға болады.

Бағдарламаның коды:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Danie_1
    {
        class Pervaj_1
        {
            bool flag;
            public void Niti_1()
            {
                Console.WriteLine("Работает нить");
                if (!flag)
                {
                    flag = true;
                    for (int i = 0; i < 10; i++)
                        Console.Write(i + " ");
                    Console.WriteLine();
                }
```

```
            }
        }
        static void Main()
        {
            Pervaj_1 tt = new Pervaj_1();
            new Thread(tt.Niti_1).Start();
            tt.Niti_1();
            Console.ReadLine();
        }
    }
}
```

Результат работы программы:
Работает нить
Работает нить
0 1 2 3 4 5 6 7 8 9
или

Работает нить
0 1 2 3 4 5 6 7 8 9
Работает нить

Біздің бағдарламаның жұмысынан көргеніміздей, екі тізбек те **flag** жергілікті айнымалысымен жұмыс істейді. Сандардың тек бір реттілігі ғана шығарылады, ал басқасы блокталынады – тізбек **flag** айнымалысының мәнін «көреді».

Егер екі объект құрып, әр объекті үшін өз тізбегін іске қосса, онда әр объекті үшін (тиісінше әр тізбекке) өзінің **flag** жергілікті айнымалысы құрылатын болады және де әртүрлі тізбектердің жергілікті айнымалыны бірлесе пайдалануы тоқтатылады.

Бағдарлама коды:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Danie_1
    {
        class Pervaj_1
        {
            bool flag;
            public void Niti_1()
            {
                Console.WriteLine("Работает нить");
                if (!flag)
                {
                    flag = true;
                    for (int i = 0; i < 10; i++)
                        Console.Write(i + " ");
                    Console.WriteLine();
                }
            }
            static void Main()
```

```
            {
                Pervaj_1 tt = new Pervaj_1();
                new Thread(tt.Niti_1).Start();
                Pervaj_1 vv = new Pervaj_1();
                new Thread(vv.Niti_1).Start();
                //tt.Niti_1();
                Console.ReadLine();
            }
        }
}
```

Работает нить
Работает нить
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
или

Работает нить
0 1 2 Работает нить
0 1 2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
и т.д.

Сонымен, түрлі тізбектердің **деректерді ортақ пайдалануы** тізбектердің тек **бір объектіге тиесілі** әдістер немесе деректерді пайдаланған кезде ғана орындалады.

Әртүрлі тізбектердің деректер мен әдістерді пайдалануы туралы теорияда, әдетте, тізбектердің «статикалық» элементтермен жұмыс істеуі қарастырылады, яғни **static** қатынау спецификаторы бар элементтермен тізбектің жұмыс істеуі сұрақтары қарастырылады. Біздің бағдарламада бір статикалық айнымалыны және бір әдісті жариялап, және де оларды әртүрлі тізбектерде қолданып көреміз.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Danie_1
 {
  static int k = 0;
  static void cikl()
  {
   k++;
   Console.WriteLine("Работает поток " + k);
   int n1, n2;
   n1 = k * 10; n2 = (k + 1) * 10;
   for (int i = n1; i < n2; i++)
        Console.Write(i + " ");
   Console.WriteLine();
  }

  static void Main()
  {
   new Thread(cikl).Start();
```

```
            new Thread(cikl).Start();
            new Thread(cikl).Start();
            new Thread(cikl).Start();
            cikl();
            Console.ReadLine();
        }
    }
}
```

Бағдарлама жұмысы:

Работает поток 1
Работает поток 3
50 Работает поток 2
50 51 52 53 54 55 56 57 58 59
50 51 52 53 54 55 56 57 58 59
51 52 53 54 55 56 57 58 59
Работает поток 4
Работает поток 5
50 51 52 53 54 55 56 57 58 59
50 51 52 53 54 55 56 57 58 59

Іске қосылған тізбектердің барлығы бір статикалық айнымалыны пайдаланады және бір әдіспен жұмыс істейді. Және де тізбектер жұмысының қауіпсіздігінің термині – тиімділікті (рентабельдідік) пайдаланса, онда олардың барлығы тиімсіз (рентабельді емес) болады.

Один из вариантов обеспечения безопасности работы нитей и использования общих данных заключается в использовании в них оператора **lock**, позволяющего блокировать работу всех остальных нитей (фактически блокирование всей программы) до завершения работы текущей нити.

Оператор **lock** относится к специальным блокирующим конструкциям языка C#.

Тізбектер жұмысының қауіпсіздігін қамтамасыз етудің және ортақ деректерді пайдаланудың бір варианты - ағымдағы тізбектің жұмысы аяқталғанға дейін барлық қалған тізбектердің жұмысын блоктауға (іс жүзінде бүкіл бағдарламаны блоктау) мүмкіндік беретін **lock** операторын пайдалану болып табылады.

lock операторы C# тілінің арнайы блоктаушы конструкцияларға жатады.

C# тілінде бағдарламалау бойынша оқулықтардың әртүрлі авторлары **lock** операторын қалай анықтайтынын қарастырайық.

lock операторы **try/finally** блогымен **Monitir** класының **Enter** және **Exit** әдістерін шақырту үшін синтаксистік қысқарту болып табылады [Албахари б.743].

lock негізгі сөзі код блогын уақыттың әр мезетінде оны тек бір тізбек қана пайдалана алатындай етіп блоктауға мүмкіндік береді [Троелсен б.312].

Павловская Т.А. болса, **lock**-ты келесі жазба форматы бар оператор ретінде анықтайды

lock (өрнек)

{ операторлардың блогы}

Өрнек блоктауды қажет ететін объектіні анықтайды. Қарапайым әдістер үшін өрнек ретінде **this** негізгі сөзі, ал статикалық әдістер үшін – **typeof(**клас) пайдаланылады.

Операторлардың блогы блоктауды қажет ететін кодтың күрделі (критический) секциясын анықтайды [Павловская б. 242].

Бұл анықтамалардың барлығында **lock** көптеген тізбектер пайдалануы үшін блокталатын кодтың қандай да бір блогын енгізеді деген бірыңғай анықтама бар. Әдістемелік көзқарастан дәрістерімізде ұстанатын Т.А. Павловскаяның анықтамасы ең жақсы және түсінікті болып табылады, дәрістерімізде әрі қарай соны қолданамыз.

Сонымен, **lock** операторы дегеніміз - ол код секциясын уақыттың әр мезетінде оны тек бір ғана тізбек пайдаланытындай етіп блоктауға мүмкіндік беретін оператор.

Бағдарламаның коды:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Danie_1
 {
  static int k = 0;
  static object t = typeof(object);
  static void cikl()
  {
   Console.WriteLine("Работает нить ---- " + k);
   lock (t)
   {
    k++;
    Console.WriteLine("Работает нить № " + k);
    int n1, n2;
    n1 = k * 10; n2 = (k + 1) * 10;
    for (int i = n1; i < n2; i++)
         Console.Write(i + " ");
    Console.WriteLine();
   }
  }

  static void Main()
  {
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   new Thread(cikl).Start();
   cikl();
   Console.ReadLine();
  }
 }
}
```

Бағдарлама жұмысы:

```
Работает нить ---- 0
```

```
Работает нить ---- 0
Работает нить ---- 0
Работает нить ---- 0
Работает нить ---- 0
Работает нить № 1
10 11 12 13 14 15 16 17 18 19
Работает нить № 2
20 21 22 23 24 25 26 27 28 29
Работает нить № 3
30 31 32 33 34 35 36 37 38 39
Работает нить № 4
40 41 42 43 44 45 46 47 48 49
Работает нить № 5
50 51 52 53 54 55 56 57 58 59
```

Біздің бағдарламада бір мезгілде барлық тізбектер іске қосылады, бірақ кезекті тізбектің орындалуы **lock** операторының көмегімен бағдарламаның қалған тізбектерінің орындалуын блоктауға әкеп соғады.

Бұл сияқты міндеттер тізбек жұмысы кезінде бағдарламаның басқа тізбектерімен де пайдаланылатын кейбір айнымалылардың мәндері тізбек жұмысының соңына дейін өзгермеу керек болғанда пайда болады. Мысалы, банк клиенттерінің шоттарымен операциялар атқарғанда.

Жұмысын аяқтаған тізбек қайтадан басынан басталмайды.

4.2.2 Тізбектерге деректерді жіберу

Тізбек объектілерін құру кезінде пайдаланылатын (жоғарыда қарастырылған) **ThreadStart** делегатында деректерді тізбектерге жіберу үшін қажетті формальді параметрлер жоқ – делегат іске қосылатын әдістің атауын ғана анықтай алады.

Бірақ, **C#** тілінде тізбек конструкторы асыра (перегружен) жүктелген және тек объектіні құруға ғана қабілетті емес, сондай-ақ **ParameterizedThreadStart** деген қандай да бір параметрді жіберетін **.NET Framework** платформасының басқа да делегатын пайдалануға рұқсат етеді. Осы делегаттың келесі жазба форматы бар:

public delegate void ParameterizedThreadStart(object obj);

ParameterizedThreadStart әдісі **object** типіндегі тек бір параметрді қабылдауға қабілетті – **C#** тіліндегі кез келген объект **object**-ден туынды болып саналады.

Пайдаланылатын делегаттың (оқу мақсаттарында) атауын анық көрсететін және әдіске қандай да бір аргументті жіберетін оқу мысалын қарастырайық.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Nit_Par
 {
  static void cikl(object ob)
  {
   int k = (int)ob;
   Console.WriteLine("Работает нить ---- " + k);
   k++;
   Console.WriteLine("Работает нить № " + k);
```

```
            for (int i = 0; i <= k; i++)
                 Console.Write(i + " ");
            Console.WriteLine();
          }

          static void Main()
          {
              Thread[] name = new Thread[5];
              for (int j = 0; j < 5; j++)
              {
               name[j] = new Thread(new
ParameterizedThreadStart(cikl));
               name[j].Start(j);
              }
             Console.ReadLine();
          }
         }
        }
```

Бағдарлама жұмысы:

Работает нить ---- 0
Работает нить № 1
0 1
Работает нить ---- 1
Работает нить ---- 2
Работает нить ---- 3
Работает нить № 4
0 1 2 Работает нить № 3
0 1 2 3
3 4
Работает нить ---- 4
Работает нить № 5
0 1 2 3 4 5
Работает нить № 2
0 1 2

Келтірілген мысалда **Thread** класының объектілерінің массивы құрылады.

Объектілер ParameterizedThreadStart(object obj) делегатының көмегімен құрылады.

Делегат тізбектің әдісі іске қосылған кезде **object** типіндегі бір параметрді жіберуге мүмкіндік береді.

Тізбекте іске қосылатын әдіс алынған параметрдің бүтін санға анық түрленуін орындауға тиіс.

Біз атап кеткендей, **C#** тілінде тізбек конструкторы асыра жүктелген және бастапқыда берілгендей, яғни өзгеріссіз **.NET Framework** платформасының бірнеше делегатын пайдалануға жол береді. Сондықтанда, әдетте бағдарламада пайдаланылатын делегаттың атауы жазылмайды, мысалы, біздің бағдарламада **Main** әдісі келесідей жазылады:

```
        static void Main()
          {
              Thread[] name = new Thread[5];
              for (int j = 0; j < 5; j++)
              {
                  name[j] = new Thread(cikl);
```

```
            name[j].Start(j);
        }

        Console.ReadLine();
    }
```

Тізбек жұмысының реттілігін мысалы, **lock** операторының көмегімен жүргізуге болады.

Тізбектерге деректерді жіберудің басқа тәсілі, объектінің белгілі бір данасының объектісіз және оның өрістерінсіз пайдаланылатын статикалық әдісін емес, объектінің өрістерімен жұмыс істеуге мүмкіндік беретін әдісін тізбекте іске қосу болып табылады.

Объектінің таңдалған данасының қасиеттері тізбектердің тәртібін анықтайтын болады.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  public static int kol=0;
  class T
  {
   private double data;
   private int c, n;
   public Thread thread1;

   public T()
   {
    thread1 = new Thread(run);
    thread1.Start(kol);
   }
   public void run(object ob)
   {
    lock (this)
    {
     kol++; n = kol;
     Console.WriteLine("Нить {0} стартовала ",kol);
     Console.WriteLine("Введите число");
     c = int.Parse(Console.ReadLine());
if (kol == 4) data = c*40;
     if (kol < 4)
     {
      switch (kol)
      {
       case 1: data = Math.Cos(c); break;
       case 2: data = Math.Exp(c); break;
       case 3: data = c + 30; break;
      }
      T thr2 = new T();
      thr2.thread1.Join();
     }
```

```
      Console.WriteLine("Нить {0} завершена! Значение равно:
{1}",n,data);
     }
    }
   }
   static void Main()
   {
    T thr1 = new T();
    thr1.thread1.Join();
    Console.ReadKey();
   }
  }
}
```

Бағдарлама жұмысы:
Нить 1 стартовала
Введите число
1
Нить 2 стартовала
Введите число
2
Нить 3 стартовала
Введите число
3
Нить 4 стартовала
Введите число
4
Нить 4 завершена! Значение равно: 160
Нить 3 завершена! Значение равно: 33
Нить 2 завершена! Значение равно: 7,38905609893065
Нить 1 завершена! Значение равно: 0,54030230586814

Біздің мысалда 4 тізбек іске қосылады, әрі олардың әрқайсысы **T** класының **run** әдісін пайдаланады. Параметр ретінде мәні қосылған тізбектің нөміріне сәйкес келетін **kol** айнымалысын алады. **Run** әдісі **T** класының әдісі болып табылады, сондықтанда ол осы кластың объектілерінің өрістерімен жұмыс істей алады, мысалы, әр тізбек үшін жеке берілген **c** өрісінің мәнін пайдаланып, қандай да бір өрнекті есептейді. Біздің бағдарламада әр тізбекте нәтижесі **data** нақты типті айнымалыға берілетін өз өрнегі қолданылған.

Бағдарламада кезекті тізбекті қосудың рекурсивтік тәсілі пайдаланылған – іске қосылатын тізбектердің санын шектеу if шартының көмегімен жүргізіледі (kol < 4). **kol** айнымалысы 0-ден бастап өзгереді. Әр қосылған тізбекте өзінің T объектісі құрылады thr2 = new T().

Құрылған объектінің ішінде шақыртқан объектіні оның аяқталуына дейін блоктайтын Join() (thr2.thread1.Join();) әдісі іске қосылады.

Көп тізбекті қосымшаларды пайдаланудың тиімділігі туралы - Интернеттен шолу (тізбекті кейді ағындар деп атайды).

Ағындарды қашан пайдаланған жөн

Егерде орындалатын тапсырма көп уақыт алатын болса, онда көп ағындылықтың маңыздылығы болады, өйткені басқа компьютерден жауап тосу қажет, (қосымшаның сервері, деректер базасының немесе клиенттің сервері).

Көп ағынды типтік қосымша ұзақ есептеулерді фондық режімде атқарады. Басты ағын орындауды жалғастырады, сол уақытта жұмыс ағыны фондық тапсырманы орындайды.

Windows Forms қосымшаларында басты ағын ұзақ есептеулерді жүргізіп жатқанда, ол пернетақта мен тінтуірдің хабарламаларын өңдей алмайды және қосымша еш белгі бермейді. Осы себеп бойынша *"Жұмыс істеп тұрмын... Мархабат, тосыңыз"* деген жазбамен басты ағын пайдаланушыға модалдық диалогты көрсетсе де, жұмыс ағынында көп уақыт алатын тапсырмаларды іске қосу қажет, өйткені бағдарлама ағымдағы операция аяқталмаса, келесі операцияға ауыса алмайды. Осындай шешім, пайдаланушыны үдерісті үзуге итермелейтіндей, операциялық жүйе қосымшаны "Жауап бермейтін" деп белгілемейтіндігіне кепілдік береді. Осы жағдайда да модалдық диалог «Болдырмау» («Отмена») батырмасын ұсынуы мүмкін, өйткені осы форма тапсырма фондық ағында орындалып жатқанда, хабардарды алып жатады.

Бір ағынды web-сервер тек қана нашар емес, ол жай ғана мүмкін емес! Күйлерді (stateless) сақтамайтын қосымшалар серверінің жағдайында, көп ағындылық әдетте қарапайым түрде жүзеге асырылатыны қуантады. Статикалық айнымалылардағы деректерге қатынауды синхрондауда қиындықтар туындауы мүмкін.

Ағындар қашан қажет емес

Көп ағындылықтың артықшылықтармен қатар, кемшіліктері де бар. Олардың ең бастысы – бағдарламалардың күрделілігін біршама арттыру. Күрделілікті қосымша ағындар емес, олардың өзара әрекеттесуін ұйымдастыру қажеттілігі арттырады. Әзірлеу циклының ұзақтығы, сондай-ақ бағдарламада кездейсоқ (анда-санда) пайда болатын және қиын анықталатын қателердің саны осы өзара әрекеттесу қаншалықты ниетті болғандығына байланысты болады. Сонымен, ағындардың өзара әрекеттестігінің дизайнын қарапайым етіп ұстау керек, немесе сіз кодты қайта жазуға немесе реттеуге деген табиғилыққа қарсы қабілеттілігіңіз болмаса, көп ағындылықты мүлдем пайдаланбау қажет.

4.3 Тізбектердің жұмыс режимдері. Windows-тағы үдерістер

4.3.1 Негізгі және фондық тізбек жөніндегі түсінік

Бастапқыда тізбектер негізгілер ретінде құрастырылады, бұл осындай тізбектердің бірі орындалып жатқанда қосымша аяқталмайтындығын білдіреді. C# тілінде фондық тізбектер деген түсінік бар, олар барлық негізгі тізбектер аяқталған соң, сол мезгілде аяқталады (олар қосымшаның өмірін «ұзартпайды» деп саналады).

Тізбек мәртебесі - негізгі немесе фондық, логикалық типі бар IsBackground қасиетінің күйімен анықталады.

Оқу бағдарламасы ретінде кідіртудің әртүрлі уақыты бар екі тізбектің жұмысын қарастырайық:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  public static void Niti_1()
  {
   Console.WriteLine("Работает нить 1 :");
   for (int i = 0; i < 10; i++)
```

```
        {
         Console.Write(i + " ");
         Thread.Sleep(1000);
        }
        Console.WriteLine();
       }
       public static void Niti_2()
       {
        Console.WriteLine("Работает нить 2 :");
        for (int i = 10; i < 20; i++)
        {
         Console.Write(i + " ");
         Thread.Sleep(1000*2);
        }
        Console.WriteLine();
       }
       static void Main(string[] args)
       {
        Thread nt1 = new Thread(Niti_1);
        Thread nt2 = new Thread(Niti_2);
        nt1.IsBackground = false;
        nt2.IsBackground = true;
        //nt1.IsBackground = true;
        //nt2.IsBackground = false;
        nt1.Start();
        nt2.Start();
        Console.ReadLine();
       }
      }
     }
```

Рабрта программ:
Работает нить 1 :
0 Работает нить 2 :
10 1 11 2 3 12 4 5 13 6 7 14 8 9 15
16 17 18 19

Работает нить 1 :
0 Работает нить 2 :
10 1 11 2 3 12 4 5 13 6 7 14 8 9

Бірінші жағдайда `nt1.IsBackground = true;` болғанда – бірінші тізбек фондық, бағдарлама екінші тізбек жұмысы аяқталғанда өз жұмысын аяқтайды. Екінші жағдайда (екінші тізбек фондық болады) бағдарлама өз жұмысын екінші тізбек жұмысын аяқтағанға дейін аяқтайды. «Фондық ағын осындай тәсілмен аяқталғанда, ағын ішіндегі барлық **finally** блоктары ескерілмейді. **Finally**-дағы кодтың орындалмауы әдетте қолайсыз болғандықтан, қажетті таймаутты тағайындап (**Thread.Join** көмегімен), бағдарламадан шығар алдында барлық фондық ағындардың аяқталуын тоқсан дұрыс болады. Егер қандай да бір себептер бойынша жұмыс ағыны берілген уақытта аяқталмаса, онда оны апаттық түрде аяқтап көруге болады (**Thread.Abort**).

Жұмыс ағынын фондық ағынға ауысу қосымшаны аяқтаудың соңғы мүмкіндігі болады, өйткені сөнуші /***умирающий***/ негізгі ағын қосымшаға аяқталуға мүмкіндік бермейді. Тұрып қалған негізгі ағын WindowsForms қосымшаларында әсіресе аса қауіпті, өйткені

қосымша оның басты ағыны аяқталғанда аяқталады (пайдаланушы үшін), бірақ үдеріс орындалуды жалғаса береді. Тапсырмалар диспетчерінде оның орындалатын файлының атауы орындалып жатқан үдерістердің тізімінде қалса да, ол тапсырмалар диспетчерінде қосымшалар тізімінен жоғалып кетеді. Пайдаланушы оны тауып тоқтатқанша /***прибьет***/ дейін, үдеріс ресурстарды тұтынуын жалғастырады және де қосымшаның қайтадан қосылған данасының қосылуына немесе қалыпты қызмет етуіне кедергі жасайды.»

4.3.2 Тізбектің басымдылығы туралы түсінік

Тізбектің сипаттамаларын анықтағанда, тізбек басымдылығын Priority қасиетінің көмегімен беруге болады. C# тілінде тізбекке беруге болатын Lowest, BelowNormal, Normal, AboveNormal, Highest деген басымдылықтың 5 градациясы бар. Тізбектің басымдылығы тізбекке басқа тізбектерге қарағанда қаншама уақыт берілетінімен анықталады. Тізбектің басымдылықтарын зерттеу үшін екі негізгі тізбектері бар, бірақ әртүрлі басымдылықтармен алдыңғы бағдарламаны пайдаланамыз.

Бағдарлама коды:

```
using System;
using System.Diagnostics;
using System.Threading;

namespace ConsoleApplication1
{   class Program
 {     public static void Niti_1()
  {      Console.WriteLine("Работает нить 1 :");
   for (int i = 0; i < 10; i++)
   {      Console.Write(i + " ");
   }
   Console.WriteLine();
  }
  public static void Niti_2()
  {
   Console.WriteLine("Работает нить 2 :");
   for (int i = 10; i < 20; i++)
   {
    Console.Write(i + " ");
   }
   Console.WriteLine();
  }
  static void Main(string[] args)
  {
   Thread nt1 = new Thread(Niti_1);
   Thread nt2 = new Thread(Niti_2);
   nt1.Priority = ThreadPriority.BelowNormal;
   nt2.Priority = ThreadPriority.BelowNormal;
   nt1.Start();
   nt2.Start();
   Console.ReadLine();
  }
 }
}
```

Тізбек басымдылығының мәні әртүрлі болғандағы бағдарлама жұмысы:

А)

```
nt1.Priority = ThreadPriority.AboveNormal;
nt2.Priority = ThreadPriority.Normal;
```

Работает нить 1 :
Работает нить 2 :
10 11 12 13 14 15 16 17 18 19
0 1 2 3 4 5 6 7 8 9
B)

```
nt1.Priority = ThreadPriority.Highest;
nt2.Priority = ThreadPriority.Lowest;
```

Работает нить 1 :
0 1 2 3 4 5 6 7 8 9
Работает нить 2 :
10 11 12 13 14 15 16 17 18 19
C)

```
nt1.Priority = ThreadPriority.BelowNormal;
nt2.Priority = ThreadPriority.BelowNormal;
```

Работает нить 2 :
10 11 12 13 Работает нить 1 :
0 1 2 3 4 5 14 15 16 17 18 19
6 7 8 9

Тізбектің басымдылығымен орындалу уақытының қатынасы қандай?

4.3.3 Процесс түсінігі

Windows-та процесстер **ядро объектісі** деп аталады. Ядро объектісіне ағымдағы қосымшада қолданылатын жүйелік ресурстар тиісті.

Әр процесстің орындалуы басты тізбектің орындалуынан басталады. Консольдік қосымшалар үшін ол **Main** әдісі болып табылады.

Процесстің орындалуы барысында оның басқа тізбектері немесе басқа процесстер де құрылуы мүмкін.

Процесс жұмысы оның барлық тізбектерінің жұмысы аяқталғанда аяқталады.

Windows-та әр процесс келесі жүйелік ресурстарға ие:

- виртуалды адрестік кеңестік;
-қауіпсіздік жүйесінің ақпараты бар қол жеткізу/қатынау маркері;
- ядро объектілерінің дескрипторларын сақтауға арналған кесте.

Тізбектерге ұқсаса әр үдерісте қызметтік бағдарламалар пайдаланылатын өз **идентификаторы** болады.

4.3.4 Жұмыс істеп тұрған қосымшада процессті құру

Жұмыс жасап тұрған қосымшадан қосымшаны іске қосу үшін (үдерісті құру) **Process** класын пайдалану қажет.

Process класында (осы дәрістің қосымшасында ұсынылған) қасиеттер мен әдістердің жиынтығы бар.

Тізім орта анықтамасының көмегімен `Process` сөзін анықтап, F1 батырмасын басқанда алынған.

Қосымшаға процессті сәтті қосу үшін қосымшаға **System.Diagnostics** атаулар кеңестігін қосу қажет.

Процесстерді қосу үшін **Process** класын және **Process.Start(FileName) немесе Process.Start()** әдістерін қолдану керек.

Process.Start(FileName) әдісін іске қосу алдында әдіс параметрі ретінде процесс файлының атауын немесе оның толық анықтау керек.

Жаңа процесс объектісін құруға болады (FileName қаисетіне мән беріледі және параметрлерсіз Process.Start() әдісінің көмегімен процессті іске қосуға болады).

Ұсынылған іске қосудың нұсқалары келесі бағдарламада жүзеге асырылған:

```
using System;
using System.Diagnostics;
namespace ConsoleApplication1
{
 class soz_pro
 {
  static void Main()
  {
   Process.Start("sol.exe");
   Console.ReadLine();
   Process myProcess = new Process();
   myProcess.StartInfo.FileName = "Notepad";
   myProcess.Start();
   Console.ReadLine();
  }
 }
}
```

Бағдарламаның жұмысының мәнісі «ПасьянсКосынка» ойынын қосу. Ойын соңында «Enter» пернесін басып «Блокнот» редакторын қосу керек. Блокнотпен жұмыс аяқталған соң «Enter» пернесіне басумен консольдік терезені жабу керек.

Process класының әдістерін пайдалануға болады, мысалы, GetProcesses жұмыс істеп тұрған үдерістің немесе үдерістердің сипаттамаларын зерделеу үшін.

Келесі оқу бағдарламасында ағымдағы мезетке жұмыс істеп тұрған бірінші үдерістің кейбір сипаттамалары қарастырылады:

```
using System;
using System.Diagnostics;
using System.ComponentModel;

namespace ConsoleApplication1
{
 class soz_pro
 {
  static void Main()
  {
   foreach (Process p in Process.GetProcesses())
   {
    Console.WriteLine("Name:         " + p.ProcessName);
    Console.WriteLine("PID:          " + p.Id);
    Console.WriteLine("Started:      " + p.StartTime);
    Console.WriteLine("Memory:       " + p.WorkingSet64);
    Console.WriteLine("CPU time:     " +
p.TotalProcessorTime);
    Console.WriteLine("Threads:      " + p.Threads.Count);
    Console.WriteLine(); break;
   }
   Console.ReadLine();
  }
 }
}
```

Бағдарлама жұмысы:

Name: dwm
PID: 1292
Started: 31.07.2011 7:07:47
Memory: 31358976
CPU time: 00:00:03.2604209
Threads: 6

5 C# ТІЛІНДЕ ТІЗБЕКТЕРДІҢ ЖҰМЫСТАРЫН СИНХРОНИЗАЦИЯЛАУ

5.1 Тізбектерді синхронизациялау

5.1.1 Әрекет түсінігі

Тізбек контекстін өзгертетін командалардың кез келген реттілігі **әрекет** (действие) деп аталады.

Тізбек контексті - бұл тізбек өзінің орындалуы кезінде жүгіне алатын компьютер жадысының облысы (анықтама 2 дәрісте қарастырылады).

Әрекет өзінің орындалуы кезінде үзілмесе, және әрекеттің өзімен ғана өзгертілсе,онда **әрекет үзіліссіз (үздіксіз)** деп аталады.

Әрекет тек микропроцессордың үзу сигналымен (сигналом прерывания) ғана үзілуі мүмкін.

Бірінші талап тек бірпроцессорлық жүйелерге ғана әділ болады, оларда әрекеттің орындау уақытына үзуге де тыйым салуға болады.

Мультипроцессорлық жүйелер үшін әрекеттер әртүрлі тізбектерде параллельді орындалуы да және де бір-бірімен қиылысуы да мүмкін.

Сондықтанда мультипроцессорлық жүйелер үшін **әрекеттердің үзіліссіздігі** бір процессорда орындалатын әрекеттің басқа процессорда орындалып жатқан әрекетті өзгертуге тыйым салумен қамтамасыз етіледі.

Сонымен, мультипроцессорлық жүйелерде әрекеттердің үзіліссіздігін қамтамасыз етудің маңызды міндеті - параллельді тізбектер жұмысының келістіру (үйлестіру, согласование).

Осы міндетті орындау нұсқаларының бірі **тізбектердің жұмысын синхронддау**.

Жалпы алғанда, **тізбектерді синхрондау дегеніміз** – тізбектер өзара алмасатын сигналдардың көмегімен оларды орындаудың қандай да бір бекітілген реттілігіне қол жеткізу

Синхрондау дегеніміз – болжанылған нәтижені алу үшін тізбектер әрекеттерін координациялау [Албахари

5.1.2 Тізбектерді синхрондау құралдарының жіктелуі

Осы дәрісте бір үдерісте тізбектерді синхрондау мысаларындағы синхрондаудың теориялық аспектілерін қарастырамыз.

Әдетте, бағдарламаның негізгі әдісі (Main) басты тізбек деп аталады, ал басты тізбек үшін кейбір міндеттерді шешетін әдістер қосымша немесе екінші тізбектер деп аталады.

Әрине, осы тізбектердің жұмысы реттелуге тиіс. Тізбектер жұмысының реттелгендігінен басқа реалды (айқын) (оқуға емес) бағдарламалар кейбір деректерді бірлесе пайдалану міндеттерін шешуге тиіс (тізбектерге деректерді және тізбектер арасында деректерді жіберудің жеке сұрақтарын біз алдыңғы дәрістерде қарастырдық), бірақ синхрондау теорияларын зерделеу үшін осы міндеттерді жеке-жеке қарастырған тиімді.

«Барлық синхрондау құралдарын шартты түрде төр санатқа бөлуге болады:

- тізбектерді синхрондаудың қарапайым құралдары, мұнда глобальді (ауқымды) айнымалылар және тізбектің орындалуын тоқтатудың қарапайым әдістері (Sleep және Join) жатады;
- арнайы блоктаушы конструкциялар,мұнда синхрондау объектілері (Mutex және Semaphore) және lock операторы жатады;
- арнайы сигналдарды беруге және қабылдап алуға негізделген конструкциялар. Осы конструкциялар басқа тізбектен сигнал алғанға дейін тізбектің орындалуын тоқтатып тұрады. Әдетте, сигналдық механизмді пайдаланатын екі осындай конструкциялар қолданылады, олар оқиғаларды күту дескрипторлары және Monitor класының Wait/Pulse әдістері;
- тізбектердің орындалуын тоқтатуды талап етпейтін құралдар (бөгет етпейтін (кідіртпейтін) синхрондау құралдары), оларға Interlocked класы және volatile арнайы кілт сөзі, ол деректерді оқу –жазу операцияларын кэштеуге тыйым салуға мүмкіндік береді.»[Албахари б. 739-740].

Тізбектерді синхрондаудың аталған құралдарынан басқа, ContextBoundObject класымен және Synchronization атрибутымен жүзеге асырылатын тізбектің ресурстарына қатынауды автоматты синхрондау (блоктау) нұсқасы бар. Оларды біз бесінші санат құралдары деп атаймыз.

Синхрондаудың қандай да бір құралдарын таңдау шешілетін міндетпен анықталады және бағдарламашыға байланысты.

Синхронда құрадларының зертелуін ең қарапайым санаттардан бастаймыз.

Албахари Интернетте жарияланған мақаларының бірінде тізбектерді синхрондау құралдарының келесі тізімін келтіреді (мақала аудармашысы оларды ағындар деп атайды):

«Тізбектерді синхрондаудың маңызды құралдары»

Келесі кестелерде ағындарды кординациялау (синхрондау) үшін .NET инструменттары келтірілген:

Бұғаттаудың қарапайым әдістері

Конструкция	**Тағайындалуы**
Sleep	Көрсетілген уақытта бұғаттау
Join	Басқа ағынның аяқталуын күту

*Бұғаттаушы конструкции*лар

Конструкция	**Тағайындалуы**	**Басқа**	**Жылдамдығы**

		үдерістен қолжетімді ме??	
Lock	Ресурсқа немесе код секциясына тек бір ғана ағын қол жеткізе алатындығына кепілдік береді. Уақыттың әр мезетінде оны тек бір ғана тізбек қолдана алатындай етіп, код секциясын блоктайды	Жоқ	Жылдам
Mutex	Ресурсқа немесе код секциясына тек бір ғана ағын қол жеткізе алатындығына кепілдік береді . қосымшаның бірнеше данасын қатар іске қосуды болдырмас үшін қолданылуы мүмкін.	Ия	Орташа
Semaphore	Ресурсқа немесе код секциясына қол жеткізе алатын ағындар саны көсетілгеннен аспайтындығына көрсетілген ғана кепілдік береді.	Ия	Орташа

(автоматты түрде бұғаттау үшін синхронизация контекстері де қолданылуы мүмкін)

Конструкция	Тағайындалуы	Басқа үдерістен қолжетімді ме?	Жылдамдығы
EventWaitHandle	Ағынға басқа ағыннан сигнал күтуге мүмкіндік береді.	Ия	Орташа
Wait and Pulse*	Ағынға бұғаттаудың берген шарты аяқталғанға дейін күтуге мүмкіндік береді	Жоқ	Средне

Сигналдық конструкциялар

Конструкция	Тағайындалуы	Басқа үдерістен қолжетімді ме?	Жылдамдығы
Interlocked*	Қарапайым бұғаттамайтын атомарлы операцияларды орындау.	Ия – бөлінісілетін жады арқылы	Аса жоғары
volatile*	Өрістерге қауіпсіз бұғатталмайтын қатынау үшін	Да – бөлінісілетін жады арқылы	Аса жоғары

5.1.3 Тізбектерді синхрондаудың қарапайым құралдары

Тізбектерді синхрондаудың теорияларын қарастыру кезінде, дәстүрлі түрде олардың жұмысының кезектілігін көзбен бақылау үшін кейбір әртүрлі мәндердің циклдық шығарылуы қолданылады, мысалы, әртүрлі символдар немесе сандарды шығару. Дәстүрден алшақтамайық, біздің тізбектердің жұмысы да монитор экранның консолдық терезесіне сандық мәндерді шығаруда болады (реалды тізбектер басты тізбек үшін жұмыс атқарады).

Басты тізбектен басқа, біздің бағдарламада қосымша екі тізбек бар деп болжайық. Басты тізбек консолдық терезеге 0-ден 9-ға дейін, бірінші қосымша тізбек –10-нан 19-ға дейін, ал екінші қосымша тізбек –20-дан 2-ға дейін сандарды шығарсын (олардың жұмысы осындай).

Тізбектері синхрондалмаған, глобальді айнымалының көмегімен және бір үдеріс тізбектерін синхрондау құралдары - **Sleep, Join**и **lock** әдістерінің көмегімен синхрондалған бірнеше оқу бағдарламаларын қарастырайық.

Синхрондалмаған тізбектері бар бағдарламада бірінші тізбек өз класының ішінде жарияланған, ал екінші тізбек – статикалық әдіспен көрсетілген.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Not_Cinxr
    {
        class Pervaj_1
        {
            public void Niti_1()
            {
                for (int i = 10; i < 20; i++)
                    Console.Write(i + " ");
            }
        }
        static void Niti_2()
        {
            for (int i = 20; i < 30; i++)
                Console.Write(i + " ");
        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
            for (int i = 0; i < 10; i++)
                Console.Write(i + " ");
            Console.ReadLine();
        }
    }
}
```

Бағдарлама жұмысы:
0 20 21 10 11 12 13 14 15 16 17 18 19 1 2 3 4 5 6 7 8 9 22 23 24 25 26 27 28 29
Тізбектердің жұмысы синхрондалмаған – шығару «араласып» жүргізіледі.

5.1.3.1 Ауқымды айнымалыны пайдалану

Flag глобальді айнымалысының көмегімен тізбектерді синхрондау мысалын қарастырайық. Барлық тізбектер жұмысқа қосылады, бірақ тізбектердің «денелері» ауқымды айнымалыдан тізбек жұмысына рұқсат «тосатын» **goto** және **if** операторларымен блокталынады.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Cin_Gl
    {
        public static int flag = 0;
        class Pervaj_1
        {
            public void Niti_1()
            {
               m1: if (flag == 0)
                 {
                     for (int i = 10; i < 20; i++)
                         Console.Write(i + " ");
                     flag++;
                 } else goto m1;
            }
        }
        static void Niti_2()
        {
          m1:  if (flag == 2)
            {
                for (int i = 20; i < 30; i++)
                    Console.Write(i + " ");
                flag++;
            } else goto m1;
        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
        m1: if (flag == 1)
            {
                for (int i = 0; i < 10; i++)
                    Console.Write(i + " ");
                flag++;
            }
            else goto m1;
            Console.ReadLine();
        }
```

```
        }
    }
```

Бағдарлама жұмысы:

10 11 12 13 14 15 16 17 18 19 0 1 2 3 4 5 6 7 8 9 20 21 22 23 24 25 26 27 28 29

Алдымен (**Flag** = 0) бірінші тізбегі жұмыс істейді, кейін (**Flag** = 1) басты тізбегі және (**Flag** = 2) екінші тізбегі жұмыс істейді.

5.1.3.2 Sleep әдісін пайдалану

Ағымдағы тізбекті берілген миллисекундқа блоктайтын Thread.Sleep() әдісінің көмегімен тізбектерді синхрондау мысалын қарастырайық.

Thread.Sleep(0) әдісі – бөлінген уақыт квантынан бас тартады.

Thread.Sleep(100) әдісі –тізбектің орындалуын 100 миллисекундқа блоктайды –» 100 миллисекундқа «ұйықтау».

```
Thread.Sleep(0); // отказаться от одного кванта времени CPU
  Thread.Sleep(1000);         // заснуть на 1000 миллисекунд
  Thread.Sleep(TimeSpan.FromHours(1));  // заснуть на 1 час
  Thread.Sleep(Timeout.Infinite); // заснуть до прерывания
```

Тізбек блоктаушы конструкцияларын пайдаланған соң блоктау аяқталғанша дейін уақыт кванттарын алуын тоқтататынын атап кету керек.

Мысалда бағдарламаның консолдық терезесіне әр шығудан кейін тізбектердің блокталуы жүзеге асырылған.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Cin_sleep
    {
        public static int flag = 0;
        class Pervaj_1
        {
            public void Niti_1()
            {
                for (int i = 10; i < 20; i++)
               { Console.Write(i + " "); Thread.Sleep(1); }
            }
        }
        static void Niti_2()
        {
                for (int i = 20; i < 30; i++)
               { Console.Write(i + " "); Thread.Sleep(1); }

        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
```

```
            Thread t2 = new Thread(Niti_2);
            t2.Start();
            for (int i = 0; i < 10; i++)
            { Console.Write(i + " "); Thread.Sleep(1); }

            Console.ReadLine();
        }
    }
}
```

Бағдарлама жұмысы:
0 20 10 21 1 11 2 12 22 13 23 3 4 24 14 25 5 15 6 26 16 27 17 7 18 28 8 19 9 29

Бағдарламаны тексеру барысында Sleep() әдісінің кідірту уақыты 1-ден 150 миллисекундқа дейін өзгерді. Бағдарлама жұмысының барлық нәтижелерінде шығару «үштікпен» жүргізіледі, бірақ «үштік» ішінде кезек сақталмады. Сонымен, тізбек кезекті шығарудан соң,басқа тізбектің жұмысына (тек бір шығаруға) рұқсат беріп қандай да бір уақытқа «ұйықтайды».

5.1.3.3 Join әдісін пайдалану

Ағымдағы тізбектің аяқталуына дейін шақырушы тізбекті блоктайтын Join() әдісінің көмегімен тізбектерді синхрондаудың мысалын қарастырамыз. Сонымен, басқа тізбек аяқталғанға дейін Join() әдісімен блоктап тастауға болады.

Join() әдісін пайдаланудың үш нұсқасы бар:

	Атауы	Сипаттамасы
	Join()	COM және SendMessage стандартты хабарламаларын жіберуді жалғастыра отыра, ағын аяқталғанға дейін шақыртушы ағынды бұғаттайды.
	Join(Int32)	COM және SendMessage стандартты хабарламаларын жіберуді жалғастыра отыра, ағын аяқталғанға дейін немесе көрсетілген уақыт аяқталғанға дейін шақыртушы ағынды бұғаттайды.
	Join(TimeSpan)	COM және SendMessage стандартты хабарламаларын жіберуді жалғастыра отыра, ағын аяқталғанға дейін немесе көрсетілген уақыт аяқталғанға дейін шақыртушы ағынды бұғаттайды.

Мысалда Join() әдісімен блоктаудың түрлі нұсқалары жүзеге асырылған («//» сәйкес әдістерді қосу немесе өшіру).

```
using System;
using System.Threading;
```

```
namespace ConsoleApplication1
{
    class Cin_sleep
    {
        public static int flag = 0;
        class Pervaj_1
        {
            public void Niti_1()
            {
              for (int i = 10; i < 20; i++)
              { Console.Write(i + " "); Thread.Sleep(10); }
            }
        }
        static void Niti_2()
        {
              for (int i = 20; i < 30; i++)
              { Console.Write(i + " "); Thread.Sleep(10); }

        }
        static void Main()
        {
            Pervaj_1 Pe = new Pervaj_1();
            Thread t1 = new Thread(Pe.Niti_1);
            t1.Start();
            t1.Join();
            Thread t2 = new Thread(Niti_2);
            t2.Start();
            t2.Join();
            for (int i = 0; i < 10; i++)
            { Console.Write(i + " "); Thread.Sleep(10); }

            Console.ReadLine();
        }
    }
}
```

Бағдарлама жұмысы – тек **`t1.Join();`**:
10 11 12 13 14 15 16 17 18 19 0 20 1 21 22 2 23 3 4 24 25 5 6 26 27 7 28 8 9 29
Бағдарлама жұмысы – тек **`t2.Join();`**:
10 20 11 21 12 22 13 23 14 24 15 25 16 26 17 27 18 28 29 19 0 1 2 3 4 5 6 7 8 9
Бағдарлама жұмысы – және **`t1.Join();`** және **`t2.Join();`**:
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 0 1 2 3 4 5 6 7 8 9

Бірінші жағдайда бірінші тізбек екінші және басты тізбектердің орындалуын «блоктайды», олар блоктау аяқталған соң синхронды емес болып жұмыс істейді.

Екінші жағдайда тек басты тізбек «блокталынады», ал бірінші және екінші тізбектер синхронды емес болып жұмыс істейді.

Үшінші жағдайда басында бірінші тізбек екінші және бастапқы тізбектердің орындалуын «блоктайды», бұдан соң екінші тізбек басты тізбекті «блоктайды».

Блоктаудың осы құралы тізбектердің жұмысын ретке келтіруге мүмкіндік береді.

5.1.3.4 lock операторын пайдалану

Lock операторы **Mutex** және **Semaphore** синхрондау объектілерімен бірге арнайы блоктаушы конструкцияларға жатады**.**

Әртүрлі үдерістердің тізбектерін блоктайтын **Mutex** және **Semaphore** синхрондау объектілеріне қарағанда, **lock** операторы тек бір үдерістің ішінде ғана тізбектердің жұмысын блоктауға мүмкіндік береді. Сондықтанда **lock** операторының жұмысын осы дәрісте қарастырамыз.

Алдыңғы дәрісте біз тізбектерге деректерді жіберу үшін **lock** операторының жұмысын қарастырған болатынбыз. Осы мысалда y = a*x + b/x + c өрнегін есептеу үшін үш тізбекті қосамыз. Бірінші тізбек c мәнін алады және тиісінше екінші және үшінші тізбектерде орындалатын a*x және b/x есептеулердің нәтижелерін тосады. x айнымалысы бағдарламада глобальді айнымалы ретінде жарияланады.

Оқу мақсатында бүкіл бағдарламаның жүзеге асырылуын 4 файл ретінде орындаймыз – бағдарлама файлы және тізбектер іске қосылатын әдістер үшін болатын кластар файлынан үш файл болады. Бағдарламаның бірінші файлының бастапқы коды келесідей түрде болады:

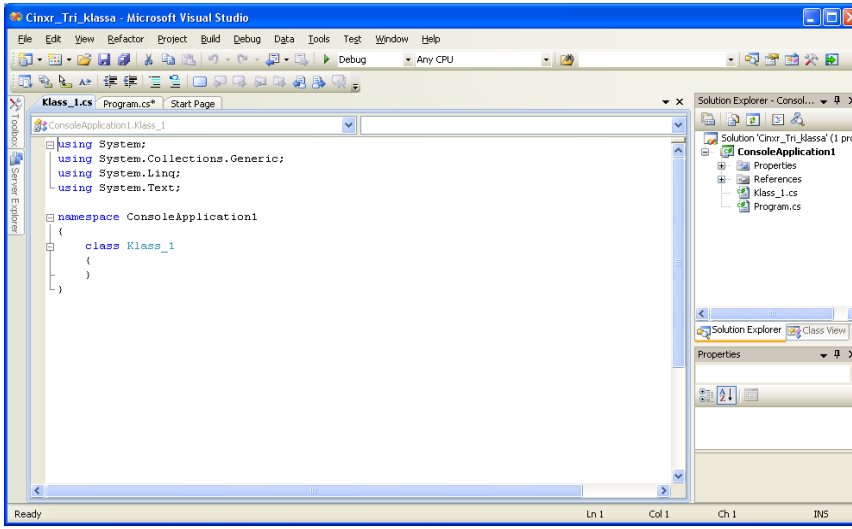
```
using System;
using System.Threading;

namespace ConsoleApplication1
{
    class Program
    {
        public static double rez,x;

        static void Main()
        {
            Console.WriteLine("Введите x");
            x = double.Parse(Console.ReadLine());
            Klass_1 Thr1 = new Klass_1();
            Thr1.thread1.Join();

            Console.ReadKey();
        }
    }
}
```

Жобаға Klass_1 файлын құру және оны қосу үшін Project режімінде AddClass командасын таңдап алу және пайда болған терезеде Klass_1 деген құрылатын файлдың атауын көрсету қажет. Орта Klass_1.cs файлын құрады, оны жобаға қосады және файл бағдарламасының кодын редакциялау режіміне ауыстырады (10.1 суретін қараңыз).



Сурет 5.1 – Klass_1.cs файлын құру және жобаға қосу

Klass_1.cs файлында диалог режиміне **c** айнымалысының мәнін енгіземіз және жұмысқа жобаның екінші тізбегін қосамыз, оның жұмыс нәтижесін бүкіл өрнекті есептеу үшін пайдаланатын боламыз.

Klass_1.cs файлының бастапқы коды:

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
 class Klass_1
 {
  private double c;
  public Thread thread1;
  public Klass_1()
  {
   thread1 = new Thread(Rab);
   thread1.Start();
  }
  public void Rab()
  {
   lock (this)
   {
    Console.WriteLine("Нить 1 стартовала");
    Console.WriteLine("Введите C");
    c = double.Parse(Console.ReadLine());
    Klass_2 Thr2 = new Klass_2();
    Thr2.thread2.Join();
    Program.rez = Program.rez + c;
```

```
            Console.WriteLine("Нить 1 завершила работу. Результат
=" + Program.rez);
           }
          }
         }
        }
```

Кодтан бірінші тізбектің жұмысын аяқтау үшін екінші тізбек жұмысының (екінші кластың әдісімен жұмыс істейтін) нәтижесі қажетті екенін көреміз. Klass_2.cs файлын құрамыз және оны жобаға қосамыз.

Klass_2.cs файлында диалог режиміне b айнымалысының мәнін енгіземіз, b/x есептейміз және жұмысқа жобаның үшінші тізбегін қосамыз, оның жұмыс нәтижесін бүкіл өрнекті есептеу үшін пайдаланамыз.

Klass_2.cs файлы келесі бастапқы кодқа ие:

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
 class Klass_2
 {
  private double b;
  public Thread thread2;
  public Klass_2()
  {
   thread2 = new Thread(Rab);
   thread2.Start();
  }
  public void Rab()
  {
   lock (this)
   {
    Console.WriteLine("Нить 2 стартовала");
    Console.WriteLine("Введите B");
    b = double.Parse(Console.ReadLine());
    Klass_3 Thr3 = new Klass_3();
    Thr3.thread3.Join();
    Program.rez = Program.rez + b/Program.x;
    Console.WriteLine("Нить 2 завершила работу. Результат
=" + Program.rez);
   }
  }
 }
}
```

Klass_3.cs файлында диалог режиміне a айнымалысының мәнін енгіземіз, a*x есептейміз және жобаның үшінші тізбегінің жұмысын аяқтаймыз. Оның жұмысының аралық нәтижесін монитор экранына шығарамыз.

```
using System;
using System.Threading;
namespace ConsoleApplication1
{
 class Klass_3
```

```
        {
         private double a;
         public Thread thread3;
         public Klass_3()
         {
          thread3 = new Thread(Rab);
          thread3.Start();
         }
         public void Rab()
         {
          lock (this)
          {
           Console.WriteLine("Нить 3 стартовала");
           Console.WriteLine("Введите A");
           a = double.Parse(Console.ReadLine());
           Program.rez = Program.rez + a * Program.x;
           Console.WriteLine("Нить 3 завершила работу. Результат
=" + Program.rez);
          }
         }
        }
```

Бағдарлама жұмысы:

```
Введите x
2,5
Нить 1 стартовала
Введите C
3
Нить 2 стартовала
Введите B
5
Нить 3 стартовала
Введите A
3
Нить 3 завершила работу! Значение равно: 7,5
Нить 2 завершила работу! Значение равно: 9,5
Нить 1 завершила работу! Значение равно: 12,5
```

Бағдарлама жұмысының нәтижелері бойынша x айнымалысының мәні енгізілген соң бірінші тізбек қосылатыны, бұдан соң екінші тізбек және кейіннен үшінші тізбек қосылатынын көреміз. Тізбектер жұмысының аяқталуы кері ретпен жүргізіледі. Тізбектер жұмысын синхрондау lock операторының көмегімен және кезекті тізбекті алдыңғы тізбектің тізбектік функциясынан қосумен қол жеткізіледі.

5.2 Арнайы блоктаушы конструкциялар

5.2.1 Mutex арнайы блоктаушы конструкция

Анықтама бойынша **Mutex** класы **lock** операторы сияқты ресурсқа немесе бір тізбектің код секциясына қатынауға рұқсат береді. Егерде тізбектерді синхрондау бір үдерісте орындалса, онда олардың арасында функционалдық айырмашылықтар жоқ. Бірақ,

Mutex класы үшін объекті құру қажет. Сондай-ақ **Mutex** объектісі **lock** операторына қарағанда баяу жұмыс істейді деп саналады.

Mutex класының негізі артықшылығы – оның объектісі әртүрлі үдерістердің тізбектерін синхрондау немесе қосымшаның бірнеше даналарын қосуын тоқтату үшін пайдалану үшін қолданылады. Ол әртүрлі үдерістерде орындалатын параллельді тізбектер арасында «өзара шығарып тастау» проблемаларын шешеді деп айтылады.

Кез келген клас сияқты **Mutex** класы да қасиеттер мен әдістерге ие (дәрістің соңындағы қосымшаны қараңыз).

Mutex-ті келесі конструктордың көмегімен құруға болады:

static Mutex бағдарламадағы аты = new Mutex();
static Mutex бағдарламадағы аты = new Mutex(bool);
static Mutex бағдарламадағы аты = new Mutex(bool, "ОЖ аты");

Біріншісі атаусыз мьютексті құрайды және ағымдағы тізбекті оның иесі етіп жасайды, сондықтан да мьютекс ағымдағы тізбекпен блокталынады.

Екіншісі мьютексті құрайтын тізбек оған ие болуға (оны блоктауға)тырысатынын анықтайтын тек логикалық аргументті қабылдайды. Егерде bool типіндегі аргументке false-де орнатса, бұл мьютекс бастапқыдан блокталған – тізбекке жататынын білдіреді.

Алғашқы екі конструктор әдетте бір үдерістегі тізбектерді синхрондау үшін қолданылады. Әртүрлі үдерістердегі тізбектерді синхрондалау біз үшін қызығушылық тудыруда. Сондықтан да біз операциялық жүйе үшін **Mutex** атау беруге мүмкіндік беретін конструктордың үшінші нұсқасын пайдаланамыз. Мысалы:

static Mutex kl = new Mutex(false, "Kljsik_1");

Оқу бағдарламасы ретінде бағдарламаның консолдық терезесіне негізгі бағдарлама (сервермен) үшін 0-ден 9-ға дейінгі сандардың мәндер кестелерін және негізгі бағдарламадан қосылатын қосымша бағдарламаның (клиентпен) 10-ден 19-ға дейін сандардың кестелерін шығаруын қарастырамыз.

Негізгі бағдарламаның коды:

```
using System;
using System.Diagnostics;
using System.Threading;
using System.Runtime.InteropServices;

namespace ConsoleApplication1
{
 class soz_pro
 {
  static void Prin(int n)
  {
   Console.Write(n + " ");
   Thread.Sleep(100);
  }
  static Mutex kl = new Mutex(false, "Kljsik_1");
  static void Main()
  {
   string clientExe = "Dop_pr.exe";
   var startInfo = new ProcessStartInfo(clientExe);
 startInfo.UseShellExecute = false;
   Process p = Process.Start(startInfo);
   for (int j = 0; j < 10; j++)
   {
       kl.WaitOne();
```

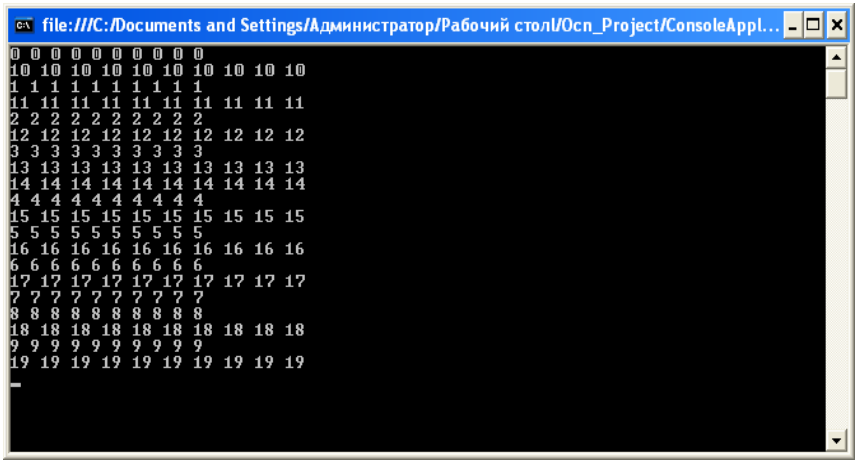
```
        for (int i = 0; i < 10; i++)
        {
            Prin(j);
        }
        Console.WriteLine();
        kl.ReleaseMutex();
    }
    Console.ReadLine();
   }
  }
}
```

Қосымша бағдарламаның коды:

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  static void Prin(int n)
  {
   Console.Write(n + " ");
   Thread.Sleep(100);
  }
  static Mutex kl_2 = new Mutex(false, "Kljsik_1");
  static void Main()
  {
   int i, j;
   for (j = 10; j < 20; j++)
   {
    kl_2.WaitOne();
    for (i = 0; i < 10; i++)
         Prin(j);
    Console.WriteLine();
    kl_2.ReleaseMutex();
   }
   Console.ReadKey();
  }
 }
}
```

Негізгі және қосымша бағдарламалардың жұмысы:



Сурет 5.2 – Негізгі және қосымша бағдарламалардың жұмысы

Бағдарламалардың қалыпты жұмысы үшін `Dop_pr.exe` файлы бағдарламалардың қалыпты жұмысы үшін негізгі бағдарламаның `ConsoleApplication1.exe`. файлымен бір папкада болуға тиіс.

Әр бағдарламаның жұмысы консолдық терезеде көрсетіледі. Бағдарламалар атауы "Kljsik_1" мьютекстің көмегімен синхрондалған – бағдарламалар ақпаратты жолдар бойынша шығарады. Жолдың үзілуіне жол берілмейді (мьютекс осы үшін бақылау жүргізеді). Кезекті жолды шығарған соң екі бағдарлама да **Mutex**ті ала алады және келесі жолдың шығаруын жалғастыра алады.

Үдеріс атауын белгілеу кезінде var типі қолданылған:

```
var startInfo = new ProcessStartInfo(clientExe);.
```

Егерде тіл компиляторы автоматты түрде инициализациялау өрнегі бойынша типті тани алса, онда C# тілінде айнымалы типіндегі осындай тапсырма мүмкін, мысалы, жолдық айнымалының инициализациялауының келесі жазбалары эквивалентті:

string st1 = “Привет”; и

var st1 = “Привет”; .

5.2.2 Semaphore арнайы блоктаушы конструкциясы

Semaphore синхрондау объектісі үдерістерді синхрондау үшін арналған келесі арнайы блоктаушы конструкция болып табылады. Негізгі оның ерекшілігі – бір үдерістің де немесе бірнеше үдерістердің де бірнеше тізбектеріне ресурсты бір мезгілде пайдалануға рұқсат береді.

Semaphore келесі конструкторлардың көмегімен құрастыруға болады:
static Semaphore бағдарламадағы аты = new Semaphore(Сч, Max);
static Semaphore бағдарламадағы аты = new Semaphore(Сч, Max, "ОЖ-гі атау");
онда Сч – Semaphore қорғалатын ресурстарды қолданатын тізбектердің есептегіші, ал Max – Semaphore арқылы ресурстарға қатынауды алуға қабілетті тізбектердің максималды саны. Әдетте бастапқы жағдайында тізбектер есептегіші семафор тізбектерінің максималды санына тең болады. Мысалы:

```
static Semaphore s = new Semaphore(3, 3);
```

Осы мысалда ресурстармен жұмысты бір мезгілде тек үш тізбекке рұқсат ететін Semaphore типіндегі s объектісі құрылады.

Semaphore-мен қорғалатын ресурстарға қатынауға рұқсат алу үшін WaitOne() әдісі қолданылады. Осы әдіс, егерде семафор сигналдық күйде болса – ресурстарға қатынауға рұқсат береді, тізбектер есептегішін бір бірлікке азайтады (егерде WaitOne() әдісінде басқа сан берілмесе). Егерде ресурстармен жұмыс істейтін тізбектердің саны семафор объектісінде белгіленген тізбектердің максималды санынан аз болса (тізбектер есептегіші нөлге тең немесе үлкен), онда семафор сигналдық күйде қалады. Ресурспен жұмыс істеуге қызығушылық танытқан тізбектер саны рұқсат берілгеннен көп бола бастаса, семафор ресурсқа қатынауды блоктайды. Release() әдісімен семафорды блоктаудан шығаруға болады. Осы әдіс семафор тізбектерінің есептегішін бір бірлікке ұлғайтады (егерде Release() әдісінде басқа сан берілмесе), ал бұл семафорды автоматты түрде сигналдық күйге ауыстырады және ол тізбектерге қол жетімді болып қалады.

Сонымен семафор объектісі бағдарлама ресурстарына көптеген тізбектердің қатынауын синхрондауға мүмкіндік береді.

Көптеген тізбектердің бағдарлама ресурстарына қатынауын синхрондау мысалы ретінде [Албахари б.748] алынған келесі бағдарламаның жұмысын қарастырамыз.

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  static Semaphore s = new Semaphore(3, 3);
  static void Main()
  {
   for (int i = 0; i < 10; i++)
       new Thread(Go).Start(i);
Console.WriteLine("Работа цикла запуска нитей закончена!");
   Console.ReadKey();
  }
  static void Go(object id)
  {
   s.WaitOne();
   Console.WriteLine(id + " <- нить начинает работу!");
   Thread.Sleep(1000 * (int)id);
   Console.WriteLine(id + " -> нить закончила работу");
```

```
            s.Release();
           }
          }
        }
```

Работа программы:

```
2 <- нить начинает работу!
4 <- нить начинает работу!
Работа цикла запуска нитей закончена!
5 <- нить начинает работу!
2 -> нить закончила работу
9 <- нить начинает работу!
4 -> нить закончила работу
0 <- нить начинает работу!
0 -> нить закончила работу
6 <- нить начинает работу!
5 -> нить закончила работу
1 <- нить начинает работу!
1 -> нить закончила работу
3 <- нить начинает работу!
3 -> нить закончила работу
8 <- нить начинает работу!
6 -> нить закончила работу
7 <- нить начинает работу!
9 -> нить закончила работу
7 -> нить закончила работу
8 -> нить закончила работу
```

Бағдарлама жұмысының нәтижесі бойынша бір мезетте бағдарлама ресурстарымен тек үш тізбек жұмыс істей алатынын көреміз. Егерде тізбектердің есептегіші және тізбектердің масималды санын бірге тең деп берсе, семафор объектісі мьютекске «ауысады». Қалған жағдайларда семафор объектісінде ұқсас объект жоқ.

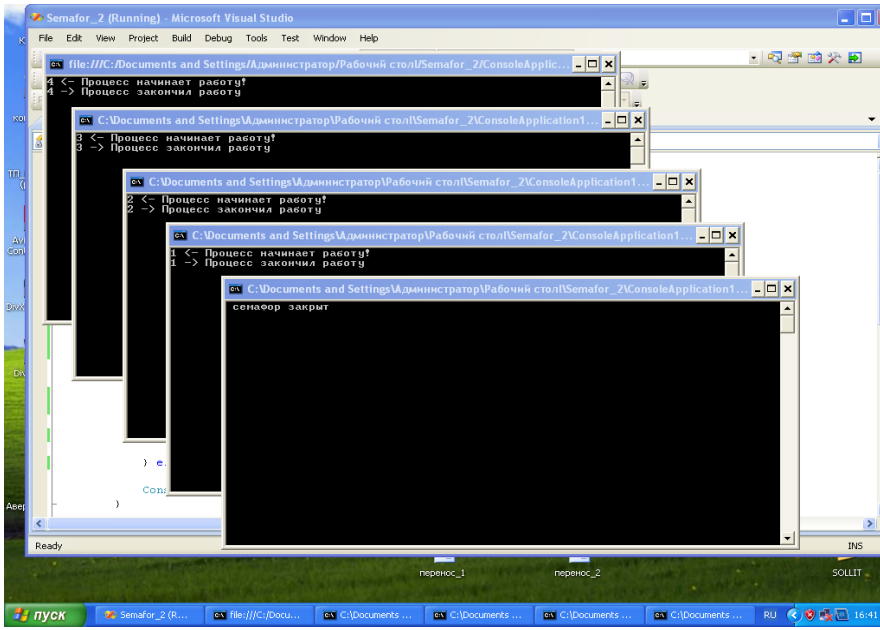
Семафор объектісінің әртүрлі үдерістермен жұмысын көрсету үшін бағдарлама өзін-өзі семафор мүмкіндіктері таусылғанға дейін қосатын таза оқу бағдарламасын пайдаланамыз.

```
using System;
using System.Diagnostics;
using System.Threading;

namespace ConsoleApplication1
{
 class Program
 {
  static Semaphore s = new Semaphore(5, 5,"sem_1");
  static void Main()
  {
   int kol;
   s.WaitOne(); kol = s.Release();
   if(kol>=1)
   {
    Process.Start("ConsoleApplication1");
    s.WaitOne();
Console.WriteLine(kol + " <- Процесс начинает работу!");
```

```
            Thread.Sleep(1000*5);
            Console.WriteLine(kol + " -> Процесс закончил работу");
            s.Release();
          } else Console.WriteLine(" семафор закрыт ");
           Console.ReadKey();
          }
        }
      }
```

Бағдарлама жұмысы:



Сурет 5.3 –Semaphore көмегімен ресурстарды үлестіру

[Албахари б.747] кітабында семафор объектісінің жұмысы бейнеленген түрде түнгі клуб жұмысымен салыстырылады.

«Semaphore түнгі клубқа ұқсас – ол дөкей қамтамасыз ететін белгілі сыйымдылыққа ие. Толған соң ешкім түнгі клубқа кіре алмайды, сыртта кезек жинақталады. Әрі қарай, бір адам клубтан кетсе, кезек басындағы біреу клубқа кіре алады. Semaphore конструкторы минимум екі параметрді қабылдайды - әлі қол жетімді орындар саны және түнгі клубтың жалпы сыйымдылығы».

5.3 Тізбектерді автоматты синхрондау

5.3.1 Тізбектерді синхрондау контекстінің түсінігі

Қандай да бір үдерістің тізбектерін синхрондау компьютердің максималды жүктелуін қамтамасыз етуге бағытталған. Көп жағдайда тізбектердің жұмысын синхрондау тізбектердің параллельді жұмысын ұйымдастырумен теңестіріледі, алайда ол олай емес.

Тізбектер бір деректерге жүгінгенде, мысалы, деректер базасымен жұмыс істегенде, тізбектердің синхрондауы айрықша маңызды болады. Тізбектердің параллельді жұмысы, мысалы, тізбектері мыңдаған сұранысты өңдейтін веб-сервердің жұмысы, ортақ деректерді қолдануды көздемейді.

Егерде объекті құрылса - ал ол қандай да бір деректермен әрекеттері бар жады облысы (**объект контексі**), және кезекті тізбектің жұмысы аяқталғанға дейін оған (**жады облысына**) деген қатынауды блоктаса, онда тізбектерді автоматты синхрондауды (нақты айтқанда блоктауды) ұйымдастаруға болады.

C# тілінде ContextBoundObject арнайы класы бар, оны мұраланған класс әдістер мен қасиеттердің белгілі жиынтығы бар **объектілер контекстерін** құрайды.

Объект контекстіне қол жеткізуді автоматты түрде **синхрондау (блоктау)** үшін осындай объектіні жариялаудың алдында **Synchronization атрибутын** көрсету қажет.

Сонда осындай объектінің **әрекет ету облысы – синхрондау контексті** деп аталады. Мысалы:

```
[Synchronization]
 public class Niti : ContextBoundObject
            { . . .}
```

Осы объектінің әрекет ету облысы **синхрондау контекст**і болып табылады.

Синхрондау контекстін құрып, осы объектінің ресурстарына бірнеше жұмыс істеп тұрған тізбектің қатынауын (блоктауын) басқаруға болады.

Сонымен үдерісте бірнеше тізбек жұмыс істеуі мүмкін, бірақ уақыттың нақты мезетінде объект ресурстарына тек бір тізбекке рұқсат етілген.

Синхрондау контекстімен жұмыс істейтін тізбек жұмысын аяқтағанға дейін, басқа тізбек қатынауға рұқсат алмайды. Бұл жайт бір мезетте бъектімен бірнеше жұмыс істеп тұрған тізбектің жұмыс жасауына жол бермейді.

5.3.2 Бірнеше тізбекпен синхрондау контекстін пайдалану мысалы

Оқу мысалы ретінде for циклының басқарушы айнымалысының мәндердін консолдық терезеге шығаруға мүмкіндік беретін әдіспен бір мезетте төрт тізбектің жұмыс істеуін қарастырамыз.

Бағдарлама коды:

```
using System;
using System.Threading;
using System.Runtime.Remoting.Contexts;

namespace ConsoleApplication1
{
[Synchronization]
 public class Niti : ContextBoundObject
 {
  public void Cikl(object ob)
```

```
        {
         int k = (int)ob;
         Console.Write("Работает нить {0}  ",k);
         for (int i = k * 10; i <= k * 10 + 9; i++)
         {
          Console.Write("  " + i);
          Thread.Sleep(100);
         }
         Console.WriteLine();
        }
       }
       class Program
       {
        static void Main(string[] args)
        {
         int i=0;
         Niti ob_1 = new Niti();
         i++;
         Console.WriteLine("Запущена нить {0}  ", i);
         Thread th1 = new Thread(new
ParameterizedThreadStart(ob_1.Cikl));
         th1.Start(i);
         i++;
         Console.WriteLine("Запущена нить {0}  ", i);
         Thread th2 = new Thread(new
ParameterizedThreadStart(ob_1.Cikl));
         th2.Start(i);
         i++;
         Console.WriteLine("Запущена нить {0}  ", i);
         Thread th3 = new Thread(new
ParameterizedThreadStart(ob_1.Cikl));
         th3.Start(i);
         i++;
         Console.WriteLine("Запущена нить {0}  ", i);
         ob_1.Cikl(i);
         Console.ReadLine();
        }
       }
      }
```

Работа программы:

```
Запущена нить 1
Запущена нить 2
Запущена нить 3
Запущена нить 4
Работает нить 2    20  21  22  23  24  25  26  27  28  29
Работает нить 1    10  11  12  13  14  15  16  17  18  19
Работает нить 4    40  41  42  43  44  45  46  47  48  49
Работает нить 3    30  31  32  33  34  35  36  37  38  39
```

Нәтижеден мысалдағы процессте барлық төрт тізбек іске қосылды, бірақ **синхрондау контекстін** қандай да бір тізбек басып алса, онда басқа тізбектің оған қол жеткізуі алдыңғы тізбектің бүкіл жұмысы аяқталған соң орындалады.

Автоматты түрдегі синхрондау **ContextBoundObject** мұрагері болмайтын (мысалы, WindowsForm) статикалық типтер мен кластардың мүшелерінің қорғанысы үшін қолданылуы мүмкін емес.

5.3.3 Өзара блоктау жөніндегі түсінік

Синхрондау контексті бір объектінің шегінен шығып таралуы мүмкін.

Бастапқыда, егер синхрондалған объект басқа объектінің кодында құрылса, екеуі де бір контекстіні бөліседі (басқаша айтқанда, бір үлкен блоктауды!).

Осы тәртіпті **Synchronization** атрибуты конструкторында **SynchronizationAttribute** класында анықталған константаларды қолдана отырып, жалаушаларды қойып, өзгертуге болады:

Константа	Мәні
NOT_SUPPORTED	Синхрондау атрибуттарын пайдаланбауға эквивалентті .
SUPPORTED	Егерде құрастыру синхрондалған объектіден болса, қолданыстағы контекстіге қосылады, әйтпесе синхрондауды қолданбау керек.
REQUIRED(default)	Егерде құрастыру синхрондалған объектіден болса, қолданыстағы контекстіге қосылу, әйтпесе жаңа контекст құру керек.
REQUIRES_NEW	Әрдайым синхрондаудың жаңа контекст құру.

Сонымен, егер **SynchronizedA** класының объектісі **SynchronizedB** класының объектісін құрса, онда оларда синхрондау контекстері әртүрлі болады. Және **SynchronizedB** төмендегідей декларацияланған болса.

```
[Synchronization(SynchronizationAttribute.REQUIRES_NEW)]
public class SynchronizedB : ContextBoundObject
{ . . . }
```

Синхрондау контексті көбірек кеңейген сайын, басқару жеңілірек болады, бірақ пайдалы параллелизм үшін мүмкіндіктер азая түсетін болады. Екінші тараптан алсақ, синхрондаудың кейбір контекстері өзара блоктауға ұшырайды. Міненки, мысал:

```
using System;
using System.Runtime.Remoting.Contexts;
using System.Threading;

namespace ConsoleApplication1
{
 [Synchronization]
 public class Niti : ContextBoundObject
 {
  public Niti ob_ni;
```

```
        public void Zapl(object ob)
        {
         int k = (int)ob;
         Console.WriteLine("Работает нить {0}  ", k);
         ob_ni.Cikl1(k);
        }
        void Cikl1(int k)
        {
         for (int i = k * 10; i <= k * 10 + 9; i++)
         {
          Console.Write("  " + i);
          Thread.Sleep(100);
         }
         Console.WriteLine();
        }
       }

       class Program
       {
        static void Main()
        {
         int i=1;
         Niti Ni_1 = new Niti();
         Niti Ni_2 = new Niti();
         Ni_1.ob_ni = Ni_2;
         Ni_2.ob_ni = Ni_1;
         Thread th1 = new Thread(new
ParameterizedThreadStart(Ni_1.Zapl));
         th1.Start(i);
         i++;
        Thread th2 = new Thread(new
ParameterizedThreadStart(Ni_2.Zapl));
         th2.Start(i);
         Console.ReadLine();
        }
       }
      }
```

Бағдарлама жұмысы:

Работает нить 1

Работает нить 2

Niti-дің әр данаса Program деген синхрондалмаған кластың ішінде құрылатындықтан, әр дана өз синхрондау контекстіне ие болады, демек, өз блоктауына да ие. Екі объект бір-бірінің әдістерін шақырғанда,бірден өзара блоктау орын алады. Контекстінің айқын блоктаудан айырмашылығы осында, оларда өзара блоктаулар бұдан гөрі айқынырақ.

Егерде `Cikl1` әдісінің кодын `Zapl` әдісінің ішіне орнатса, онда өзара блоктау мүлдем жоқ, мысалы:

```
using System;
using System.Runtime.Remoting.Contexts;
using System.Threading;

namespace ConsoleApplication1
```

```
{
 [Synchronization]
 public class Niti : ContextBoundObject
 {
  public Niti ob_ni;
  public void Zapl(object ob)
  {
   int k = (int)ob;
   Console.WriteLine("Работает нить {0}  ", k);
   for (int i = k * 10; i <= k * 10 + 9; i++)
   {
    Console.Write("  " + i);
    Thread.Sleep(100);
   }
   Console.WriteLine();
  }
 }
 class Program
 {
  static void Main()
  {
   int i=1;
   Niti Ni_1 = new Niti();
   Niti Ni_2 = new Niti();
   Ni_1.ob_ni = Ni_2;
   Ni_2.ob_ni = Ni_1;
   Thread th1 = new Thread(new
ParameterizedThreadStart(Ni_1.Zapl));
   th1.Start(i);
   i++;
   Thread th2 = new Thread(new
ParameterizedThreadStart(Ni_2.Zapl));
   th2.Start(i);
   Console.ReadLine();
  }
 }
}
```

Бағдарлама жұмысы:

```
Работает нить 1
 10Работает нить 2
 20  11  21  12  22  23  13  24  14  15  25  26  16  17  27  18  28  29  19
```

5.3.4 EventWaitHandle сигналдық конструкция

Windows жүйесінің синхрондау объектілерін құруға арналған Win32 API функцияларының жиынтығында, мьютекстер және семафорлармен қатар, **оқиғаны (event) синхрондау объектілерін құруға арналған функциялар** қолданылады. Осы синхрондау объектісінің тізбектермен жұмыс істеуі қарапайым және ыңғайлы. Ол бір тізбекті екінші тізбектің бір үдерісте болсын, әртүрлі үдерістерде болсын, қандай да бір әрекет жасағаны жайлы хабардар ету үшін пайдаланылады. Әрине, .NETFramework-та тізбектерді басқару (синхрондау) үшін осы объектінің артықшылығын пайдаланбай қоймайды. Сондықтанда жіктеу бойынша **Mutex** және **Semaphore** кейін сигналдық блоктаушы конструкциялардың келесі класы **EventWaitHandle** класының негізіндегі сигналдық конструкция болады. Сигналдық конструкция бір немесе бірнеше үдерістердің басқа тізбегінен сигналдың тосуды ұйымдастаруға мүмкіндік беретіндік береді.

EventWaitHandle класының екі туынды класы **AutoResetEvent** және **ManualResetEvent** бар (бұл бұрында қарастырылған C# оқиғалары мен делегатарына ешқандай қатысы жоқ блоктаушы сигналдық конструкция екендігін ескертеміз). Екі класқа да базалық кластың барлық функционалдық мүкіндіктері қол жетімді, жалғыз айырмашылығы – әртүрлі параметрлермен базалық кластың конструкторын шақыртуда.

«AutoResetEvent турникетке өте ұқсас – бір билет бір адамға өтуге мүмкіндік береді. “auto” деген алдынғы сөз ашық турникет біреу өтіп кеткен соң автоматты түрде жабылатындығы немесе біреуге өтуге мүмкіндік берген соң «алынып тасталынатындығы» туралы білдіреді.

Ағын **WaitOne** шақыртуымен турникет алдында блокталады (ашылмағанға дейін осы (*one*) турникет алдында тосу (*wait*).

Ал билет **Set** әдісін шақыртумен қойылады.

Егерде бірнеші ағын **WaitOne**-ні шақыртса, турникет артында кезек пайда болады. Билет кез келген ағынды, басқа сөзбен айтсақ, **AutoResetEvent** объектіге қатынауы бар кез келген (блокталмаған) ағынды «қоя алады».

Бір блокталған ағынды өткізу үшін **Set**-ті шақырта алады.

Егерде **Set** күтуші ағындар жоқ болғанда шақыртылса, хэндл қандай да бір **WaitOne**-ні шақыртқанша дейін ашық күйде болады**.** Осы ерекшелік турникетке жақындайтын ағындар арасында және билетті қыстыратын ағын (“опа, билет микросекундқа ертерек салынды, өкінішке орай, сіз қандай да бір уақыт тосуыңыз керексіз!”) арасындағы жарыстардан бас тартуға мүмкіндік береді. Бірақ, бос турникет үшін **Set** көп рет шақырту бір дегенде тұтас жиынды өткізбейді – тек бір адам өте алады, барлық қалған билеттер босқа жұмсалатын болады.

WaitOne міндетті емес **timeout** параметрді қабылдайды – егер әдіс күту сигналды алумен емес, таймаут бойынша аяқталса, онда **false**-ні қайтарады. тым көп блоктауға түспеу үшін күтуді жалғастыру (егерде втоматты блоктаумен режім қолданылса) үшін **WaitOne**-ді ағымдағы синхрондаудың контекстінен шығуға үйретуге алады.

Reset әдісі еш күтусіз және блоктаусыз ашық турникеттің жабылуын қамтамасыз етеді.

AutoResetEvent блоктаушы конструкциясын турникетпен салыстыру өте сәтті, сондықтанда интернетке сілтеме келтірілген, бірақ AutoResetEvent-тің тағы бір артықшылығын атап кету қажет – оны атаулы етуге болады. Яғни әр тізбекке өз оқиғасын «бекітуге» болады, ол осы тізбектің жұмысын шешетін болады. Әддете, set() көмегімен қосымшаны қосқанда, қандай да бір тізбектің жұмысына рұқсат беретін қандай да бір оқиға

бекітіледі. Осы тізбектің жұмысы аяқталған соң Reset-пен оқиғаны «түсіріп тастау» және set-пен келесі тізбектің жұмысына рұқсат беретін келесі оқиғаны орнату және т.б..

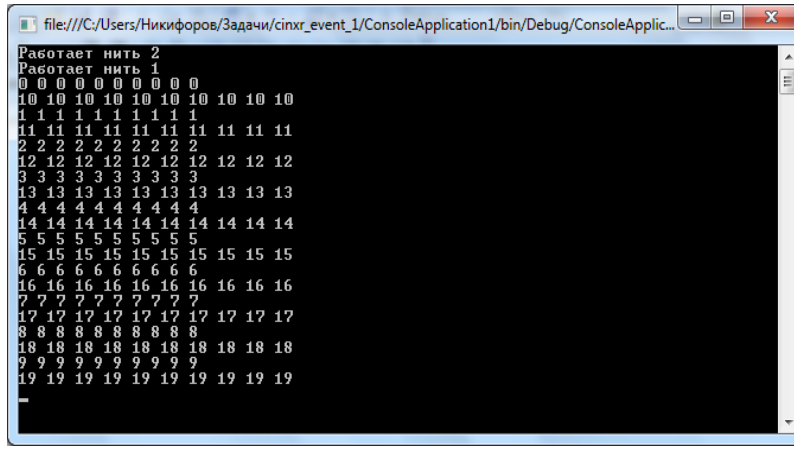
AutoResetEvent екі блоктаушы конструкциясының көмегімен екі жұптың жұмысын реттейтін оқу мысалы ретінде EventWaitHandle типінде ev1 және ev2 екі объект құрылатын бағдарлама ұсынылады. ev1 объектісі 1 тізбекті басқару, ал ev2 объектісі 2 тізбекті басқару үшін арналған. Екі тізбекте жұмысқа іске қосылған соң (олардың екеуіде өз объектілерінен рұқсатты тосады- `ev1.WaitOne();` және `ev2.WaitOne();`)`ev1.Set();`бірінші оқиға орнатылады, ол 1 тізбектің жұмысына рұқсат береді. 1 тізбектің жұмыс циклы аяқталған соң, 1 оқиға алынып тасталынады, ал 2 оқиға орнатылады және т.б.

Бағдарлама коды:

```
using System;
using System.Threading;
using System.Runtime.InteropServices;
namespace ConsoleApplication1
{
 class Program
 {
  static EventWaitHandle ev1 = new AutoResetEvent(false);
  static EventWaitHandle ev2 = new AutoResetEvent(false);
  class Pervaj_1
  {
   public void Niti_1()
   {
    Console.WriteLine("Работает нить 1 ");
    for (int i = 0; i < 10; i++)
    {
     ev1.WaitOne();
       for (int j = 0; j < 10; j++)
           Console.Write(i + " ");
       Console.WriteLine();
       ev1.Reset();
       ev2.Set();
     }    }
    public void Niti_2()
    {
     Console.WriteLine("Работает нить 2 ");
     for (int i = 10; i < 20; i++)
     {
      ev2.WaitOne();
      for (int j = 0; j < 10; j++)
           Console.Write(i + " ");
      Console.WriteLine();
      ev2.Reset();
      ev1.Set();
     }   }       }
   static void Main(string[] args)
   {
    Pervaj_1 Pe = new Pervaj_1();
    Thread t1 = new Thread(Pe.Niti_1);
    t1.Start();
    Thread t2 = new Thread(Pe.Niti_2);
    t2.Start();
```

```
            ev1.Set();
            Console.ReadLine();
           }
          }
         }
```

Бағдарлама жұмысы:



Сурет 5.4 –Тізбектерді AutoResetEvent көмегімен синхрондау

Close әдісін **WaitHandle** қажет болмай қалған соң дереу шақырту қажет - операциялық жүйенің ресурстарын босату үшін. Бірақ, егерде **WaitHandle** қосымшаның бүкіл өмір бойында қолданылса (біздің жағдайымыздағыдай), осы қадамды қалдыруға болады, өйткені ол қосымша жабылғанда автоматты түрде орындалады.