Бағдарламалауға кіріспе

1-сабақ

Мақсаты

Дағдылар / түсініктер	MTA емтиханының мақсаты
Компьютерлік бағдарламалауды түсіну	Компьютерлік қойма мен деректер түрлерін түсіну (1.1)
Шешім құрылымын түсіну	Компьютерлік шешімдер құрылымын түсіну(1.2)
Қайталану құрылымын (Repetition Structures) түсіну	Қайталануды өңдеудің тиісті әдісін анықтау (1.3)
Ерекше жағдайларды өңдеуді түсіну	Қателерді өңдеуді түсіну(1.4)

Алгоритмдер

- Алгоритм - есептерді шешудің әдісі.
- Алгоритмдерді сипаттаудың жалпы әдістері:
- блок-схема

Блок-схемалар және шешім кестелері	
Табиғи тілдерге қарағанда дәлірек	Бағдарламалау тілдеріне қарағанда аз формалды және қолдану оңай

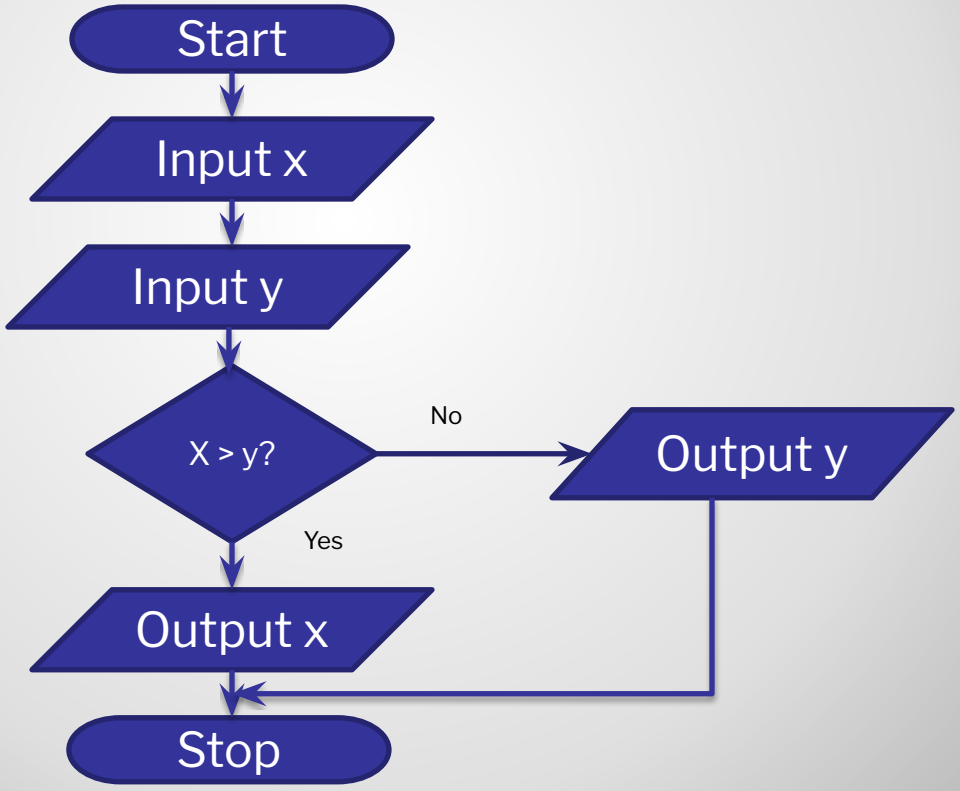
Блок-схемалар

- Графикалық схема - бұл алгоритмнің графикалық көрінісі.

Блок-схеманың жалпы белгілері	
	Алгоритмнің басталуы немесе аяқталуы
	Процесс немесе есептеу жұмысы
	Кіріс немесе шығыс операциялар
	Шешім қабылдау операциясы
	Басқару ағымының бағыты

Блок-схема мысалы

- Екі санды салыстыратын блок-схема:



Шешімдер кестесі

- шарт саны көп болғанда пайдалы
- Ықшамды және оқуға ыңғайлы формат
- Жеңілдікті есептеу туралы шешім кестесі:

Саны < 10	**Y**	**N**	**N**	**N**
Саны < 50	Y	Y	N	N
Саны < 100	Y	Y	Y	N
Жеңілдік	5%	10%	15%	20%

C # таныстыру

- Microsoft .NET Framework
  - Орындалу ортасы
  - Қайта пайдалануға болатын класс кітапханалары
  - Тіл компиляторы
- C# бағдарламалау тілі
  - .NET Framework бөлігі
  - Жоғары деңгейлі тіл
  - Бағдарлама орындалмас бұрын құрастырылуы керек.
  - Регистрге сезімтал

C # бағдарламасының құрылымы

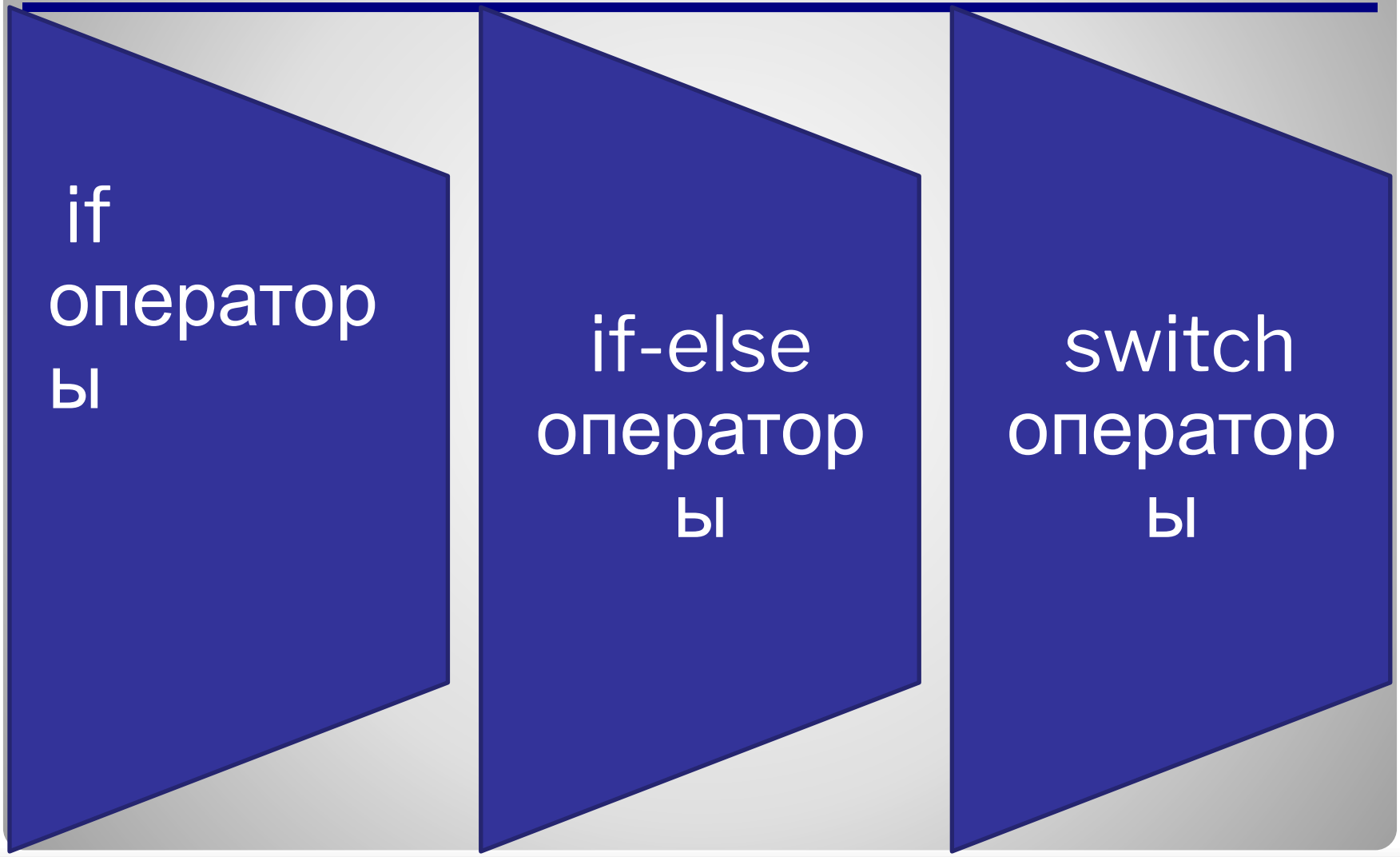
```
1  using System;
2  namespace Lesson01
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              Console.WriteLine("hello, world!");
9          }
10     }
11 }
```

C # бағдарламасының элементтері

- C # бағдарламасының жалпы элементтерін таңдаңыз:

Деректер типтері	Бағдарламадағы мәліметтер типтері. Жалпы мәліметтер типтері: int (бүтін сандар), char (символдық тип), float (өзгермелі нүктелі мәндер).
Айнымалылар	Бағдарламаны орындау кезінде уақытша сақтауды қамтамасыз етеді. int саны = 10;
Тұрақтылар	Мәні өзгертілмейтін деректер аймағы . const int i = 10;
Жиымдар	Әрбір элементке бірегей индекс арқылы қол жеткізуге болатын элементтер жиынтығы. int [] сандар = {1, 2, 3, 4, 5};
Операторлар	Нәтижені қайтармас бұрын операндтарда қандай операцияны орындау керектігін көрсететін белгілер.
Әдістер	Әдіс - бұл бірнеше операторлардан тұратын код блоктары. Әдістер кіріс мәліметтерін аргументтер арқылы алады және шақырушыға мәнді қайтара алады.

Шешім құрылымдары



if операторы

- If операторы сәйкес логикалық өрнек true болған жағдайда ғана берілген операторлар тізбегін орындайды.

```
int number1 = 10;
int number2 = 20;
if (number2 > number1)
{
    Console.WriteLine("number2 is greater than number1");
}
```

if-else операторы

- If-else операторы сіздің бағдарламаңызға логикалық өрнек true деп есептесе, бір әрекетті орындауға мүмкіндік береді, ал логикалық өрнек «false» деп есептесе, басқа әрекетті орындауға мүмкіндік береді.

```
public static void TestIfElse(int n)
{
    if (n < 10)
    {
        Console.WriteLine("n is less than 10");
    }
    else if (n < 20)
    {
        Console.WriteLine("n is less than 20");
    }
    else
    {
        Console.WriteLine("n is greater than or equal to 20");
    }
}
```

switch операторы

- switch операторы көпқадамды тармақталуға мүмкіндік береді. Көптеген жағдайларда switch операторын қолдану if-else операторларының күрделілігін жеңілдетеді

```
public static void TestSwitch(int op1, int op2, char opr)
{
    int result;
    switch (opr)
    {
        case '+':
            result = op1 + op2;
            break;
        case '-':
            result = op1 - op2;
            break;
        default:
            Console.WriteLine("Unknown Operator");
            return;
    }
    Console.WriteLine("Result: {0}", result);
    return;
}
```

Қайталану құрылымдары

- While циклы
- do-while циклы
- for циклы
- foreach циклы
- Рекурсия

while циклы

- while циклы көрсетілген логикалық өрнек жалған болғанша, операторлар блогын бірнеше рет орындайды.

```
int i = 1;
while (i <= 5)
{
    Console.WriteLine("The value of i = {0}", i);
    i++;
}
```

do-while циклы

- Do-while циклі көрсетілген логикалық өрнек жалған болғанша бірнеше рет операторлар блогын орындайды. Do-while циклі циклдің төменгі жағындағы шартты тексереді.

```
int i = 1;
do
{
    Console.WriteLine("The value of i = {0}", i);
    i++;
}
while (i <= 5);
```

for циклы

- For циклі итерацияның үш элементін - инициализация өрнегін, аяқтау шартының өрнегін және санау өрнегін - оқылатын кодқа біріктіреді.

```
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine("The value of i = {0}", i);
}
```

foreach циклы

- foreach циклы - бұл массивтер мен тізімдер сияқты топтамаларды қайталауға арналған for циклының ке

```
int[] numbers = { 1, 2, 3, 4, 5 };
foreach (int i in numbers)
{
    Console.WriteLine("The value of i = {0}", i);
}
```

Рекурсия

- Рекурсия - бұл нәтижені есептеу үшін әдіс өзін шақыруға мәжбүр ететін бағдарламалау әдісі.

```
public static int Factorial(int n)
{
    if (n == 0)
    {
        return 1;  //base case
    }
    else
    {
        return n * Factorial(n - 1);  //recursive case
    }
}
```

Ерекше жағдайларды өңдеу

- Ерекшелік - бұл бағдарламаны орындау кезінде пайда болатын күтпеген қате.
- Ерекшелік пайда болған кезде орындалу ортасы ерекшелік объектісін құрады және оны "Лақтырады".
- Егер сіз ерекше жағдайды "түсінбесеңіз", онда бағдарламаның орындалуы тоқтайды.
- Ерекшеліктер System.Exception класының немесе оның туынды кластарның бірі болып табылады:

-Мысал: DivideByZeroException ерекше нысаны бағдарлама нөлге бөлуге тырысқанда тасталады.

-Мысал: FileNotFoundException ерекшелігі - бағдарлама берілген файлды таба алмаған кезде генерацияланады.

Өңделмеген ерекше жағдайлар

- C: \ data.txt файлы осы кодта табылмаса не болады?

```
private static void ExceptionTest()
{
    StreamReader sr = null;
    sr = File.OpenText(@"c:\data.txt");
    Console.WriteLine(sr.ReadToEnd());
}
```

try-catch көмегімен ерекше жағдайларды өңдеу

- Ерекшеліктерді тудырған кодты *try* блогына салыңыз.
- Ерекшеліктерді өңдейтін кодты catch блогына салыңыз.
- Сізде әр try блогында бірнеше catch блок болуы мүмкін. Әрбір catch блогы ерекше жағдай түрін өңдейді.
- try блогы, ең болмағанда, catch блок немесе онымен байланысты finally блокты қамту керек.

Ерекшеліктерді өңдеу мысалы

```
private static void ExceptionTest()
{
    StreamReader sr = null;
    try
    {
        sr = File.OpenText(@"c:\data.txt");
        Console.WriteLine(sr.ReadToEnd());
    }
    catch (FileNotFoundException fnfe)
    {
        Console.WriteLine(fnfe.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

finally блогы

- finally блогы try блогымен бірге қолданылады.
- finally блок әрқашан ерекшелік алынып тасталғанына қарамастан орындалады.
- finally блоы тазарту кодын жазу үшін жиі қолд

```
StreamReader sr = null;
try
{
    sr = File.OpenText(@"c:\data.txt");
    Console.WriteLine(sr.ReadToEnd());
}
finally
{
    if (sr != null)
    {
        sr.Close();
    }
}
```

try-catch-finally мысалы

```
StreamReader sr = null;
try
{
    sr = File.OpenText(@"c:\data.txt");
    Console.WriteLine(sr.ReadToEnd());
}
catch (FileNotFoundException fnfe)
{
    Console.WriteLine(fnfe.Message);
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
finally
{
    if (sr != null)
    {
        sr.Close();
    }
}
```

Қайталау

- Алгоритмдер
  - Блок-схема, шешімдер кестесі
- C# бағдарламалау тілі
- Айнымалылар,Тұрақтылар,Жиымдар,Операторлар, Әдістер
- Шешімдер құрылымы
  - if, if-else, switch
- Қайталау құрылымдары
  - while, do-while, for, foreach, recursion
- Ерекше жағдайларды өңдеу
  - try-catch, try-finally, try-catch