ШАГ компьютерная академия

# ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

# Dərs №5
# C
# Dilində Proqramlaşdırma

**Mündəricat**

# 1. Daxili struktur

Keçən dərslərdə siz dövr adlı struktur ve onun C proqramlaşdırma dilində reallaşdırılması ilə tanış oldunuz. Artıq təxmin etdiyiniz kimi dövr proqramlaşdırmanın əsas strukturlarından biridir. Onun köməyi ilə çox sayda məsələ həll edilir. Həmçinin dövrün daxilinə if və switch kimi məntiqi seçim strukturlarını daxil etməyin mümkun olması ilə qarşılaşmısız. Amma əldə olunan ilə kifayətlənməyərək dövrün daxilinə ona bənzər strukturun, yəni başqa bir dövrün yerləşdirilməsinə cəhd edək. Sadə bir nümunəyə baxaq:

```
#include <iostream>
using namespace std;
void main ()
{
	int i=0,j;
	while(i<3){
		cout<<"\nOut!!!\n";
		j=0;
		while(j<3){
			cout<<"\nIn!!!\n";
			j++;
		}
		i++;
	}
	cout<<"\nEnd!!!\n";
}
```

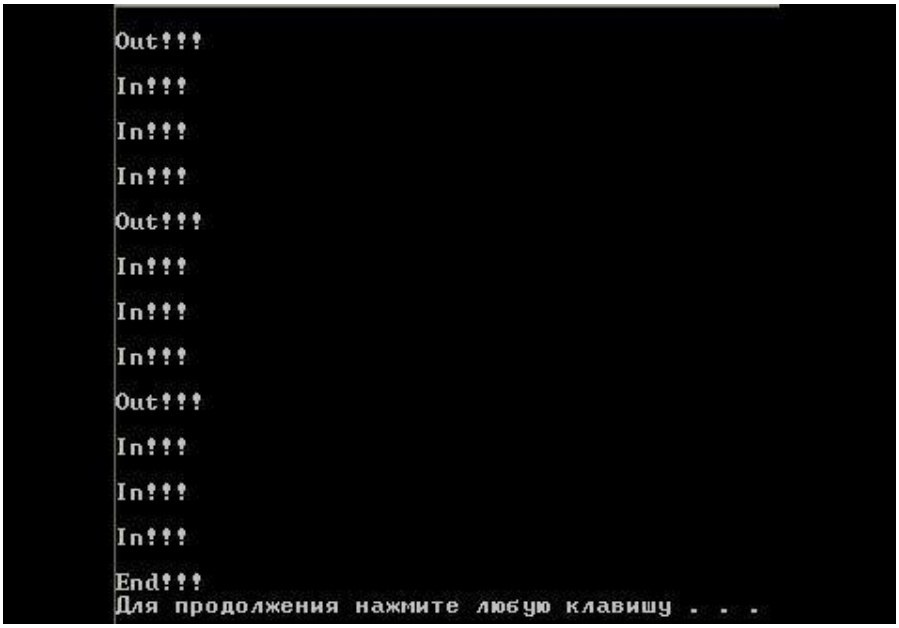
Layihənin adı NestedLoop.

Nümunəni analiz edək:

1. Proqram i<3 şərtini yoxlayır, 0-ın 3-dən kiçik olması şərti doğru olduğundan proqram xarıcı dövrə keçir.
2. Ekrana Out!!! yazısı çıxır.

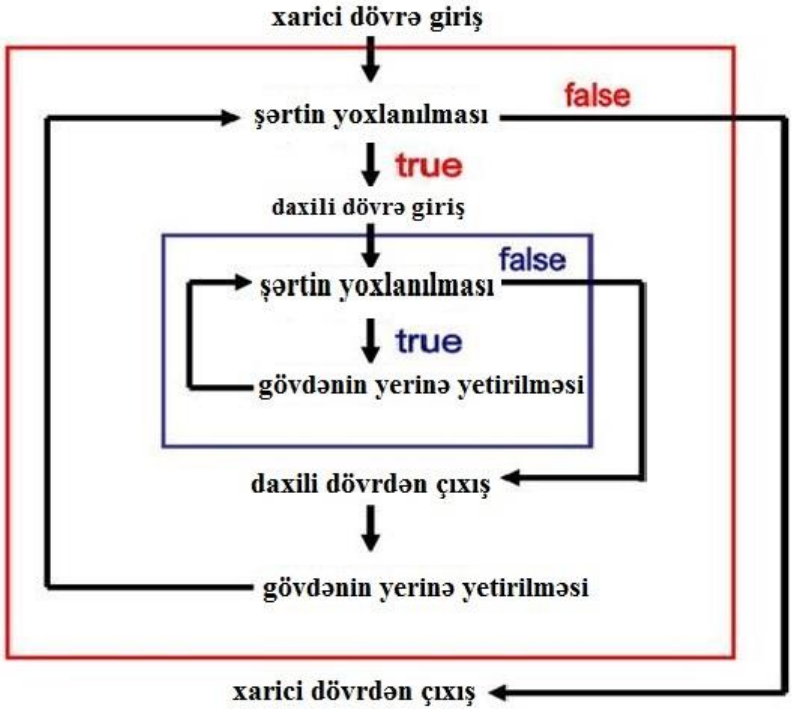
3. j dəyişəni 0-a bərabər edilir.
4. Indi j<3 şərti yoxlanılır, 0-ın 3-dən kiçik olması şərti doğru olduğundan proqram daxili dövrə keçir.
5. Ekrana In!!! yazısı çıxır.
6. İdarəedici j dəyişənin dəyişdirilməsi həyata keçirilir.
7. Yenidən j<3 şərti yoxlanılır, 1-in 3-dən kiçik olması şərti doğru olduğundan proqram daxili dövrə keçir.
8. Ekrana İn!!! yazısı çıxır.
9. İdarəedici j dəyişəninin dəyişdirilməsi həyata keçirilir.
10. Yenidən j<3 şərti yoxlanılır, 2-nin 3-dən kiçik olması şərti doğru olduğundan proqram daxili dövrə keçir.
11. Ekrana In!!! yazısı çıxır.
12. İdarəedici j dəyişəninin dəyişdirilməsi həyata keçirilir.
13. Yenidən j<3 şərti yoxlanılır, 3-ün 3-dən kiçik olması şərti yalnış sayılır və prorqam daxili dövrdən çıxır.

Sonra kod 1-ci bəndə qayıdır. Yuxarıda qeyd olunmuş bütün əməliyyatlar (1-13) 3 dəfə (i dəyişəni 3-ə bərabər olanadək) təkrar olunur. Bundan sonra proqram xarici dövrdən çıxır və ekranda End!!! yazısı görünür.




Daxil edilmiş dövrü realizə edən proqramın iş prinsipi başdan sonadək xarici dövrün hər addımında daxili dövrün tam yerinə yetirilirilməsinə əsaslanır. Başqa sözlə, nə qədər ki, proqram daxili dövrdən çıxmamışdır, xarici dövrün davam edilməsi həyata keçirilmir. Aşağıda daxili dövrlərin iş sxemi təsvir olunmuşdur.

## İş sxemi




Gördüyünüz kimi, hər şey sadədir. Amma buna baxmayaraq, daxili dövrlər bir çox çətin alqoritmlərin reallaşdırılmasını olduqca asanlaşdırır. Dərsin növbəti hissəsində veilmiş nümunələrə baxaraq buna əmin ola bilərsiz.

# 2. Praktiki nümunələr

Nümunə 1
Məsələnin quruluşu:
Vurma cedvelini ekrana çıxaran proqram yazın.
Layihənin adı MultiplicationTable.
Reallaşdırma Kodu:

```
#include <iostream>
using namespace std;
void main ()
{
	for(int i=1;i<10;i++)
	{
		for(int j=0;j<10;j++)
		{
			cout<<i*j<<"\t";
		}
		cout<<"\n\n";
	}
}
```

**Kodun şərhi:**

1. Daxili və xarici dövrün idarəedici dəyişənləri vuruq funksiyalarını yerinə yetirir.
2. i idarəedici dəyişən yaranır və ona 1 qiyməti mənimsədilir.
3. Proqram i<10 şərtini yoxlayır, 1-in 10-dan kiçik olması şərti doğru olduğundan proqram xarici dövrə keçir.
4. j idarəedici dəyişən yaranır və ona 1 qiyməti mənimsədilir.
5. Proqram j<10 şərtini yoxlayır, 1-in 10-dan kiçik olması şərti doğru olduğundan proqram daxili dövrə keçir.

6. i-nin j-ə hasili ekrana verilir - 1.
7. j idarəedici dəyişənin dəyişdirilməsi həyata keçirilir.
8. Yenidən j<10 şərti yoxlanılır, 2-nin 10-dan kiçik olması şərti doğru olduğundan proqram daxili dövrə keçir.
9. i-nin j-ə hasili ekrana verilir - 2
10. j idarəedici dəyişənin dəyişdirilməsi həyata keçirilir.

j dəyişəni 10-a bərabər olanadək 5-ci bənddən 7-dək bütün əməliyyatlar təkrarlanır. Bununla bərabər i-nin cari qiyməti (1) j-nin hər bir qiymətinə (9 da daxil olmaqla 1-dən 9-a qədər) vurulur və nəticə ekrana verilir. Beləliklə vurma cədvəlindəki 1-ə vurma sətiri alınır.

Sonra proqram daxili dövrdən çıxır və ekran kursorunu 2 sətir aşağı keçirir. Bundan sonra, i dəyişənin bir vahid artımı baş verir və yenidən daxili dövrə giriş həyata keçirilir. İndi isə 2-yə vurma zənciri ekrana cıxır.

Bu qayda ilə, axırda bütün vurma cədvəli ekrana verilir.

```
1       2       3       4       5       6       7       8       9
2       4       6       8       10      12      14      16      18
3       6       9       12      15      18      21      24      27
4       8       12      16      20      24      28      32      36
5       10      15      20      25      30      35      40      45
6       12      18      24      30      36      42      48      54
7       14      21      28      35      42      49      56      63
8       16      24      32      40      48      56      64      72
9       18      27      36      45      54      63      72      81
Для продолжения нажмите любую клавишу . . .
```

## Nümunə 2

Məsələnin qoyuluşu:

Ekrana simvollardan ibarət 20-nin 20-yə ölçülü düzbucaqlı çıxarın

Layihənin adı Rectangle.

Reallaşdırma kodu:

```
#include <iostream>
using namespace std;
void main(){
	int str;
	int star_count;
	int length=20;
	str=1;
	while(str<=length)
	{
		star_count=1;
		while(star_count<=length)
		{
			cout<<"*";
			star_count++;
		}
		cout<<"\n";
		str++;
	}
}
```

## Kodun şərhi:

1. Xarici dövrün idarəedici dəyişəni - str, düzbucaqlıdakı sətirlərin sayına nəzarət edir.

2. Daxili dövrün idarəedici dəyişəni - star_count, hər sətirdəki simvolların sayına nəzarət edir.

3. length - Düzbucaqlının tərəfinin uzunluğu.

4. Hər sətirin yazılışından sonra xarici dövrdə düzbucaqlının növbəti sətrinə keçid həyata keçirilir.

5. Nəticə belədir:


**Qeyd:** Fikir verin ki, sətirlərin sayı sətirdəki simvolların sayına uyğun olmasına baxmayaraq ekranda kvadrat alınmadı! Bu, simvolun hündürlük və eninin fərqli olmasından asılıdır.

İndi siz dövrlər, onların növləri və iş prinsipləri haqqında tam məlumata maliksiz. Amma ev tapşırığını yerinə yetirməzdən əvvəl dərsin daha bir hissəsi ilə tanış olmağınız lazımdır. Bu hissə sizə nəinki proqram yazmaqda həm də onların iş prinsiplərini analiz etməkdə kömək olacaq.

# 3. Microsoft Visual Studio birləşdirilmiş sazlayıcının istifadəsi

**Sazlama anlayışı. Sazlayıcıdan istifadənin zəruriliyi. Siz bilirsiz ki, proqramda 2 növ xəta yarana bilər.**

Kompilyasıya mərhələsindəki xəta - proqramlaşdırma dilinin sintaksisindəki xətadır. Belə xətalara kompilyator tərəfindən nəzarət olunur. Bu cür xətaya malik olan proqram işə başlaya bilməz, bu halda kodun hansı sətrində xətanın baş verməsi kompilyator tərəfindən göstəriləcək.

İcra mərhələsindəki xəta - Proqramı düzgün işləməməyə, ya da tam dayanmağa vadar edən xətadır. Qeyd etmək lazımdır ki, bu cür xəta kompilyator tərəfindən nəzarət olunmur. Yalnız nadır hallarda kompilyator hansısa düzgün olmayan təlimat barədə xəbərdarlıq verə bilər. Amma ümumiyyətlə belə hallarda proqramçı özü çıxış yolu tapmalıdır.

Məhz icra mərhələsindəki xətalardan danışacayıq. Belə bir səhvi tapmaq üçün proqramın müəyyən bir parçasını proqram yerinə yetirilirmiş kimi addim-addim keçmək lazımdır. Şubhəsiz bu halda müəyyən bir vaxtda konkret bir dəyişənin alacağı qiyməti dəqiq hesablamaq lazımdır. Bu cür proqramı sətir-sətir keçərək hesablamanı kağız üzərində aparmaq olar.

Lakin Visual Studio daxilində proqramın analızı üçün xüsusi vasitə - sazlayıcı var. Dərsin verilmiş bölməsi bu sazlayıcı ilə işin əsaslarına həsr olunur.

## Proqramın addım-addım icra edilməsi.

Tutaq ki, biz aşağıdakı kodu analiz etmək istəyirik:

```
#include <iostream>
using namespace std;
void main(){
    int sum=0;
    for(int i=1;i<15;i++){
        sum+=i;
    }
    cout<<sum;
}
```

Əvvəlcə layihə yaradın və bu kodu yığın. Onu kompilyasiya edin və əmin olun ki, sintaksis səhv yoxdur. Başlayaq, Klaviaturada F10 funksional düyməsini basın. Kodun icra edilən birinci sətri ilə yanaşı ekranda sarı ox görünəcək.

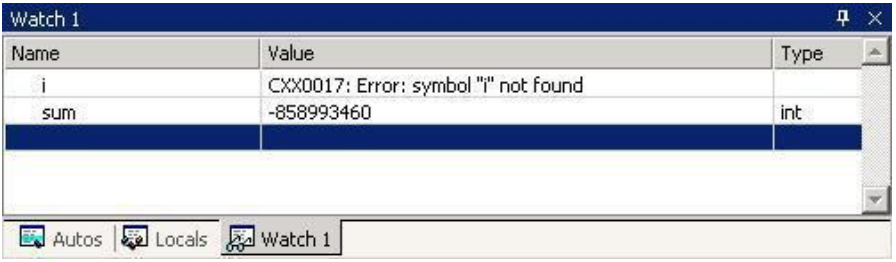
```
#include <iostream>
using namespace std;
void main(){
    int sum=0;
    for(int i=1;i<15;i++){
        sum+=i;
    }
    cout<<sum;
}
```

Məhz bu ox kodun hansı hissəsinin icra edildiyini göstərir. Növbəti addıma keçmək üçün yenidən F10 düyməsini basın. Və siz növbəti sətirə keçəcəksiniz:

```
#include <iostream>
using namespace std;
void main(){
    int sum=0;
    for(int i=1;i<15;i++){
        sum+=i;
    }
    cout<<sum;
}
```

Ekranın aşağı hissəsində dəyişənlərin analizi üçün funksiyalar yığımının olmasına fikir verin:

Autos - icra vaxtı kodun carı sətrində mövcud olan dəyişənlərin qiymətinə baxış üçün təyin olunmuşdur. Bura heç nə daxil etmək olmaz - bu avtomatik funksiyadır.




Watch - məhz elə hallar üçün təyin olunmuşdur ki, proqrama baxış üçün özümüz dəyişən seçməliyik. Siz sadəcə olaraq Name sahəsinə dəyişənin adını daxil edirsiz. Və o, yerinə yetirilən koddan asılı olmayaraq əks olunur.

İndi isə F10 düyməsini basaraq kod boyunca verilənlərin qiymətinin necə dəyişməsini izləyin.

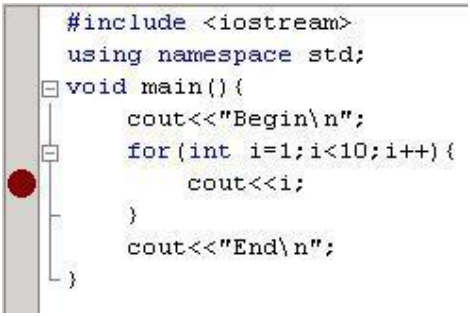
**Qeyd:** Əgər siz sazlayıcını kodun analizinin başa çatmasından daha tez dayandırmaq istəyirsinizsə Shift+F5 birləşmələrini basın.

**Qeyd:** Siz yəgin fikir verdiniz ki, sazlayıcı öz analizini proqramın birinci sətrindən başlayır. Əgər siz sazlayıcının işinin proqramın hər hansı bir sətrindən başlamasını istəyirsizsə, kursoru sizə lazım olan sətirə gətirin və Ctrl+F10 birləşməsini basın.

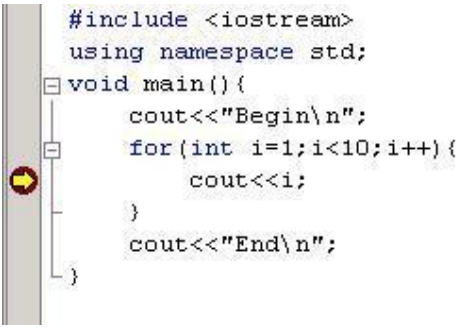
**Dayanma nöqtəsi.**

Tutaq ki, biz kodun bir hissəsini icra etmək üçün müəyyən bir yerdə dayanıb sazlayıcını işə salmalıyıq. Bunun üçün dayanma nöqtəsindən istifadə etməliyik.

Belə bir kod yığın - kursoru cout<<i; sətrinə gətirin və F9 düyməsini basın. Sətrin yanında qırmızı nöqtə əmələ gələcək.




İndi isə F5 düyməsini basın, proqram işə başlayacaq, dayanma nöqtəsinə qədər icra ediləcək və sazlayıcı rejiminə keçəcək.



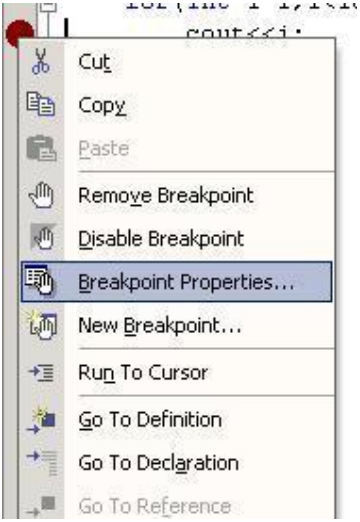

Konsolların (proqramın pəncərələri) vəziyyətinə fikir verin. Burada baş verən hər şey əks olunur:

Bundan sonra sazlayıcının işi başlayır. Sarı oxun yerini F10 vasitəsilə dəyişin və dəyişənlərlə nə baş verdiyini izləyin. Həm də konsollar pəncərəsinə nəzər salın. Kodda baş verən bütün dəyişikliklər burada əks olunur.

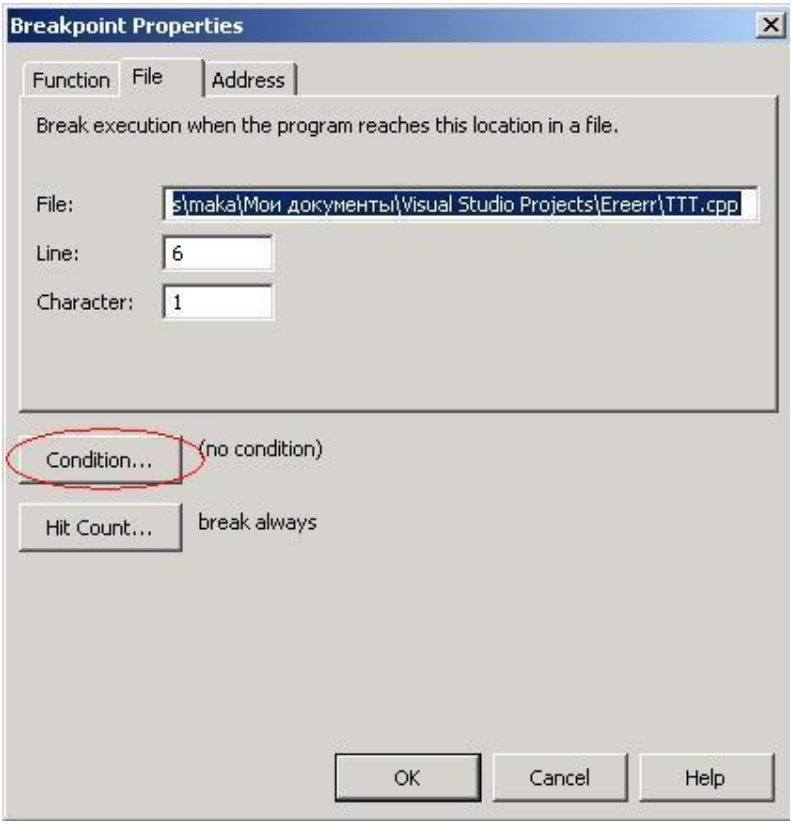
**«Ağıllı» dayanma nöqtəsi**

Biz proqramın analizini müəyyən bir nöqtədən işə saldıq. Amma qeyd edək ki, dövrün gövdəsi işə düşən kimi, hələ ilk təkrarda sazlayıcı işə başladı. Əgər təkrarlanmanın sayı çoxdursa və onların bəzilərini buraxmaq lazımdırsa, belə hallar narahatlıq yaradır. Başqa sözlə, siz analizi dövrün 5-ci təkrarından başlamaq istəyirsinizsə, dayanma nöqtəsini "ağıllı" etməklə problemi həll etmək asanddır.

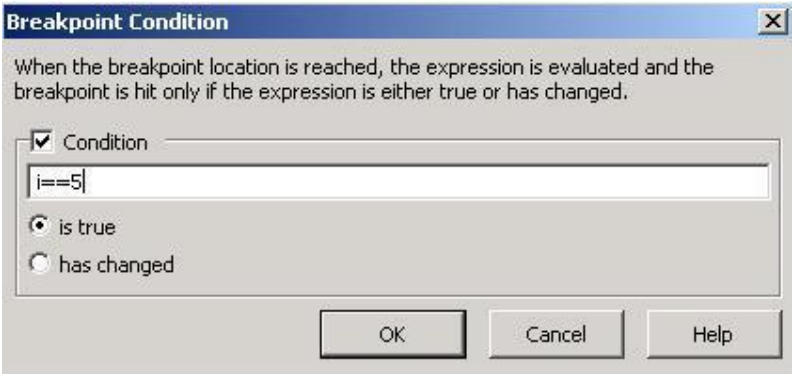
Bunun üçün həmin nöqtədə mausun sag düyməsi basılır və açılmış menyuda Breakpoint Properties seçirisiz.



Dialoq pəncərəsi açılır. File, Line və Character - dayanma nöqtəsinin qeyd olunduğu fayl, sətir və mövqedir. Bizi yanında no-condition yazılan Condition düyməsi maraqlandırır. Onu basaq.

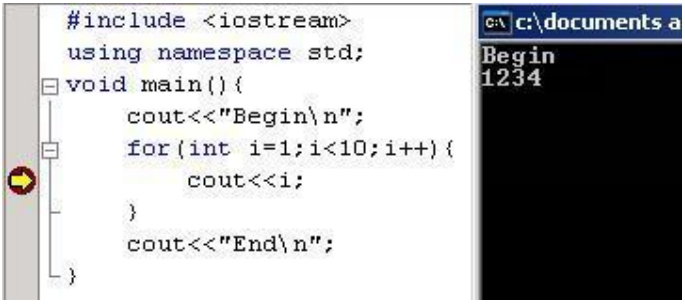



Sazlayıcının işə düşməsi şərtini yazacağımız pəncərə yarandı. Şərtimiz isə i=5-dir. Biz doğrudur cavabını seçirik, yəni şərt doğru olduğu halda dayanmaq lazımdır. OK basırıq.




İndi düymənin yanında başqa yazı göründü. Əsas pəncərədə yenə də OK seçirik.


Hər şey hazır olanda F5 düyməsini basırıq və nə baş verəcəyinə baxırıq. Gördüyünüz kimi hər şey uğurlu oldu, sazlayıcı lazimi anda işə başladı.



**Qeyd:** Dayanma nöqtəsninin yerləşdiyi sətirdə F9 düyməsini basmaqla dayanma nöqtəsni ləğv etmək olar.

Bugün biz sazlayıcının bütün imkanlarından danışmadıq. Gələn dərslərdə funskiya bölməsini öyrənəndə bu mövzuya yenidən toxunacağıq. İndi isə israrla sazlayıcı ilə işləməyi məşq etməyi tövsiyyə edirik. Məsələn, dərslərdə verilən bütün nümunələrdə və həmçinin sizin ev tapşırığınızda. Uğurlar!

# 4. Ev tapşırığı

1. 2-dən 1000-ə qədər olan bütün sadə ədədləri ekrana çıxaracaq proqram yazın. (Əgər ədəd qalıqsız olaraq yalnız özünə və 1-ə bölünürsə, onda o, sadə adlanır. 1 və 2 ədədləri sadə sayılmır).

2. Aşağıdakı fiquru ekrana çıxaran proqram yazın:

```
**********
*        *
*        *
*        *
*        *
*        *
*        *
**********
```

Fiqurun eni və hündürlüyü klaviatura vasitəsi ilə istifadəçi tərəfindən verilir.

3. Dövr vasitəsi ilə ekrana cari ayın təqvimini verin.