C ծրագրավորման լեզվի ուսուցողական ձեռնարկ

Սկսնակների համար

Տարբերակ՝ 0.1
Վերջին փոփոխութիւնը՝ 28/07/20
Էջերի քանակը՝ 29

C

Ծրագրավորման լեզու

Փաստաթղթի պատմություն		
Ամսաթիվ	**Նկարագրություն**	**Հեղինակ**
04/06/20	Ստեղծվել է փաստաթղթի առաջին տարբերակը։	Ավագ Սայան

Բովանդակություն

1. Ներածություն

1.1. Փաստաթղթի մասին

Այս փաստաթուղթը ստեղծվել է «Արմաթ» ինժեներական լաբորատորիայի տեխնիկական համայնքի կողմից։

1.2 Նպատակները

Արմաթ ինժեներական լաբորատորիաների խմբավարներին և սաներին տրամադրել C ծրագրավորման լեզվի ներկայացման հայերեն ուղեցույցը և առանձնացել դրա կարևոր հատկությունները, օպերատորները, ֆունկցիաները, հիմնաբառերն ու պարզ այլագրերը, քանի որ C-ն հանդիսանում է մեքենայական կամ ապարատուրային մակարդակի ծրագրավորման լեզու, և մեր օրերում հայտնի գրեթե բոլոր ծրագրավորման լեզուները հիմնվել են հենց C-ի վրա։

1.2. Պատմությունը և օգտագործման բնագավառները

C ծրագրավորման լեզուն ստեղծվել է 1970 թվականի սկզբներին «AT&T Bell Telephone Laboratories» ընկերությունում և այդ ժամանակվանից սկսած այն դարձել է աշխարհի ամենահայտնի և լայնորեն օգտագործված, որպես ընդհանուր նշանակության լեզու(General-purpose languages)։ C-ի օգնությամբ կարող ենք ստեղծել ինչպես պարզ հրահանգաշարային ծրագրեր, այնպես էլ ներքին ներկառուցված կոդեր, որոնց կարող ենք այսուհետ կոչել ներքին ծրագրային ապահովում (ՆԾԱ)։ ՆԾԱ-ով կարող ենք աշխատացնել փոքր միկրոբլոկներ (microcontroller), որոնք տեղադրված են տարբեր կենցաղային սարքերի մեջ ՝ հեռուստացույց, փոշեկուլ, միկրոալիքային վառարան, սառնարան, օդորակիչ, հարիչ, թեյնիկ և այլն։ Բայց C-ի միջոցով կարելի է նաև ստեղծել գրաֆիկական միջերես ունեցող ծրագրեր (գրեթե ամբողջ Unix օպերացիոն համակարգը հիմնված է C-ի վրա)։ Լեզվի հեղինակը Դենիս Ռիտչին է, որը նպատակ է ունեցել ստեղծել B լեզվին փոխարինող լեզու, ինչն էլ կդառնար որպես UNIX օպերացիոն համակարգի գրման հիմնական լեզու։ Այսպիսով, C-ն սերտ կապված է UNIX ընտանիքի օպերացիոն համակարգերին։

Հայտնի ծրագրեր, որոնք գրված են C լեզվով

Լինուքս ՕՀ-ի միջուկը, FreeBSD ՕՀ-ի միջուկը, Mathlab, Wolfram Mathematica, Oracle DB, MySQL, GCC, PostgreSQL, GIMP, Mozilla Firefox and Thunderbird, Maya 3D, KDE, JVM և այլն։

1.3. Ինչի՞ց սկսել

Լեզուն սովորելու համար նախ անհրաժեշտ է իմանալ, թե ինչ սարքեր կամ ծրագրեր են հարկավոր նախապես ունենալ։ C լեզուն կարելի է ծրագրավորել ցանկացած տեսակի համակարգչից և օպերացիոն համակարգից ՝ Windows, MacOS, Linux & Unix և այլ տեսակի ՕՀ-ում։ Ընդամենը հարկավոր է ունենալ ծրագրային խմբագրիչը՝

Windows - Visual studio code,

Linux – Terminal, Vim կամ Kdevelop,

MacOS – Terminal կամ Xcode, Brackets:

Linux/MacOS-ի դեպքում շատ պարզ է, ընդամենը բացում ենք տերմինալը և ստուգում արդյոք ունենք տեղադրված GCC (GNU Compiler Collection) կոմպիլյատոր-միջավայրը ՝ # gcc –version հրահանգի միջոցով։

Linux/Debian-ի դեպքում կարող է լինել այսպես ՝ gcc (Debian 8.3.0-6) 8.3.0, սակայն, եթե այն բացակայում է, ապա կարող եք տեղադրել հավաքելով հետևյալ հրամանը ՝ sudo apt-get install gcc։

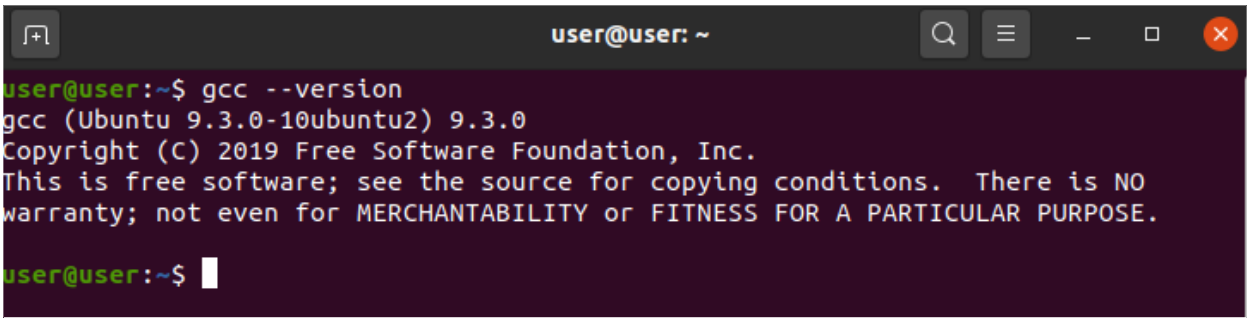
MacOS-ի դեպքում կարող է պատասխան լինել, որ Clang ծրագիրը առկա

է, որը փոխարինում է gcc-ին: Ամբողջական gcc կոմպիլյատորն ունենալու համար անհրաժեշտ է ունենալ Xcode և MacPorts, որից հետո terminal/վահանկում հավաքել հետևյալ հրամանը ՝ sudo port install gcc:

1.4. Միջավայրերը

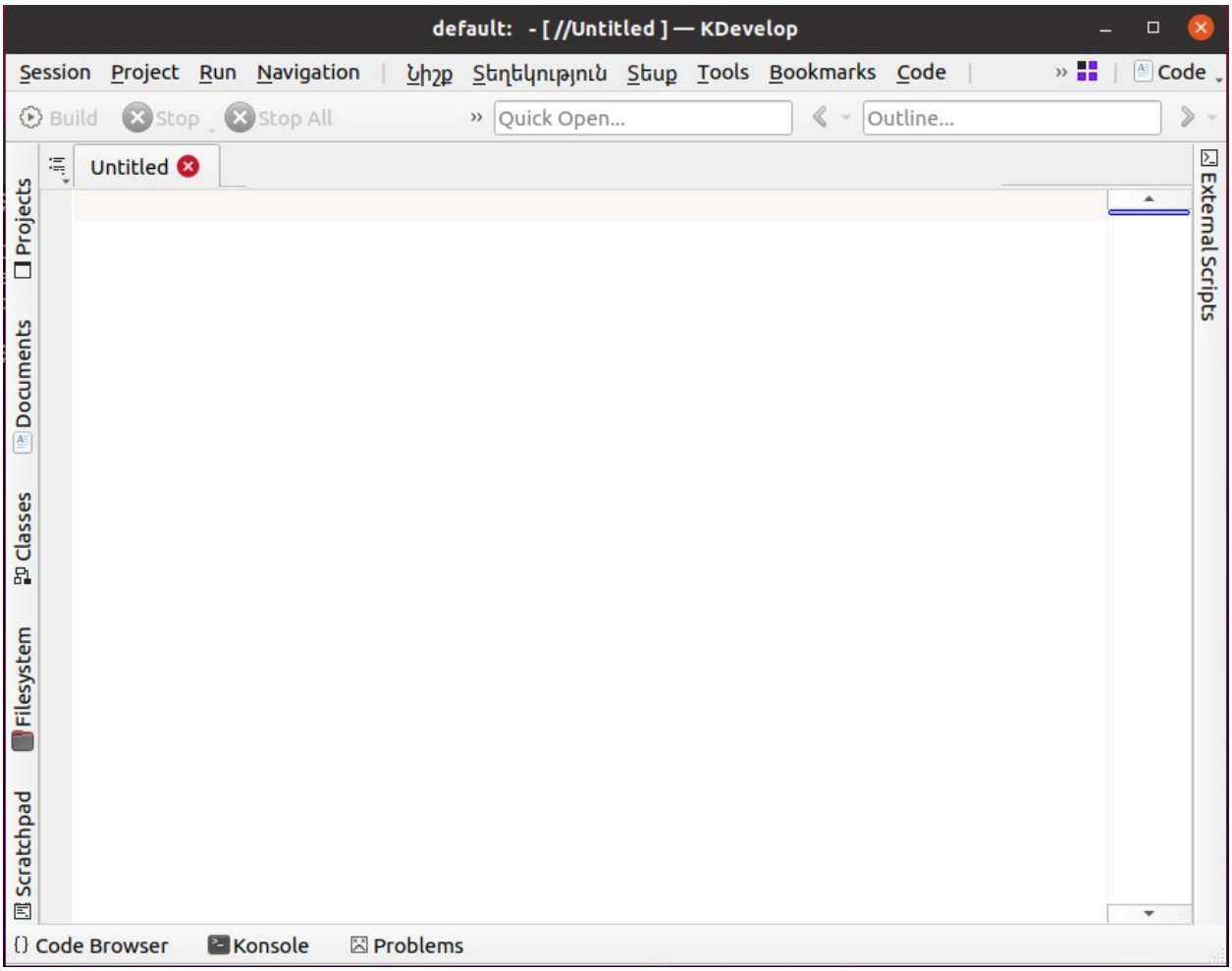
Այս ուղեցույցում հիմնականում կաշխատենք Linux/terminal, Kdevelop և Vim ծրագրերով, որոնց գրաֆիկական միջավայրերը կարող եք տեսնել կից նկարներում:

Terminal | Ubuntu



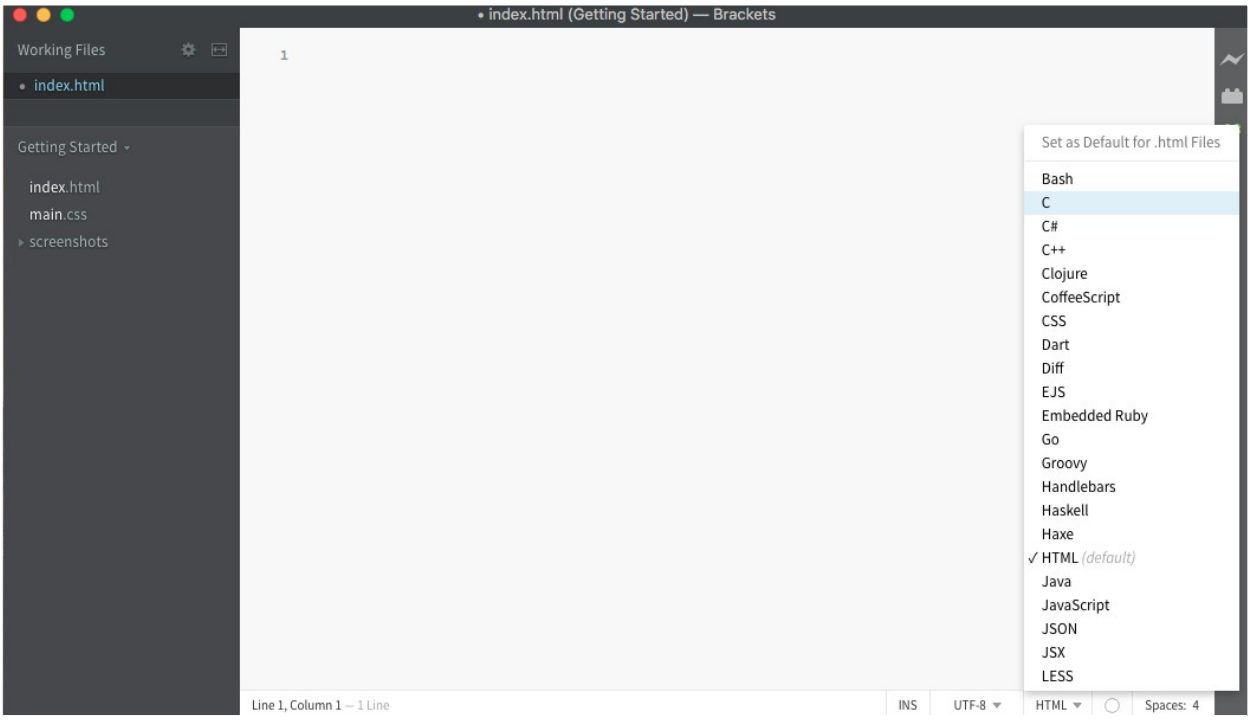
Նկար 1

Kdevelop | Ubuntu



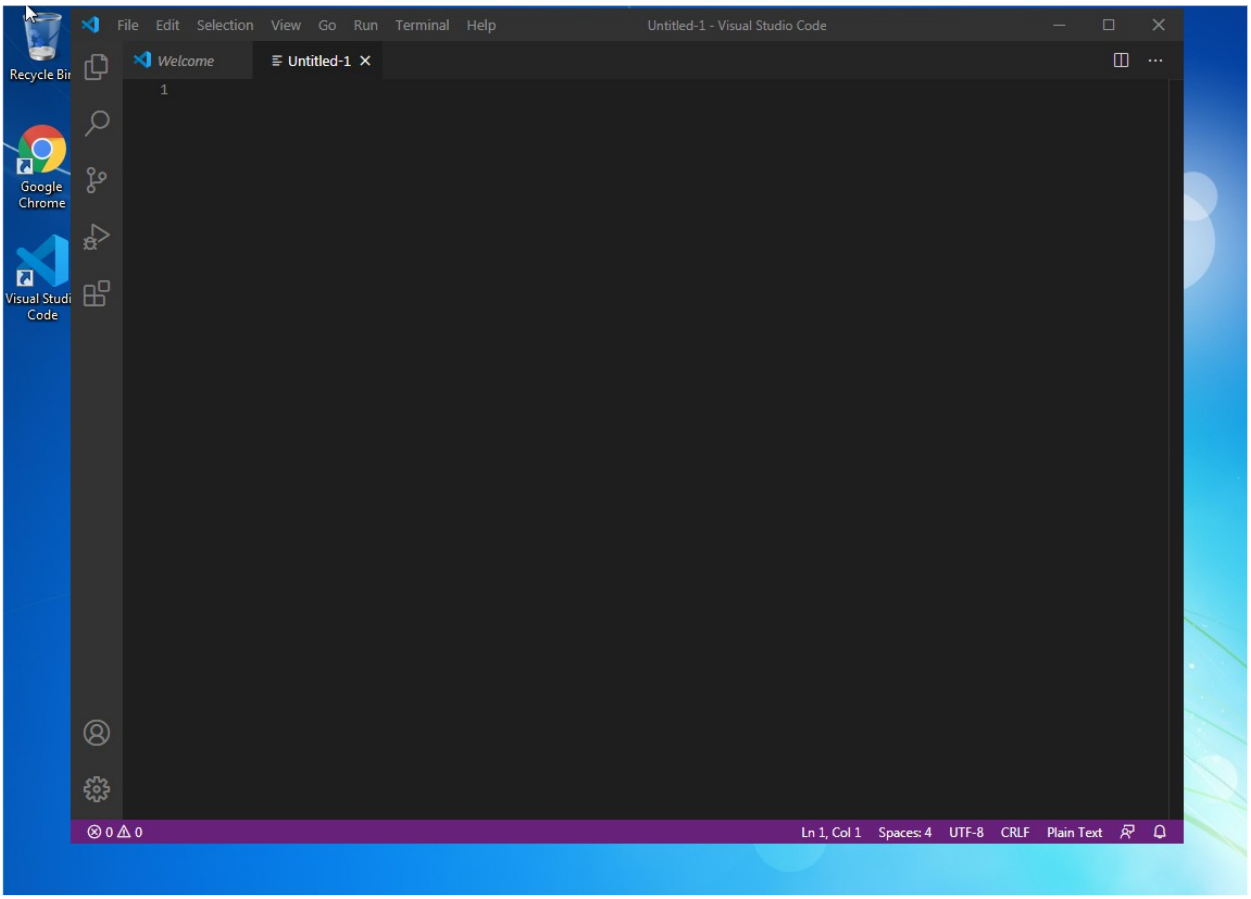
Նկար 2

Brackets | MacOS



Նկար 3

Visual Studio Code | Windows

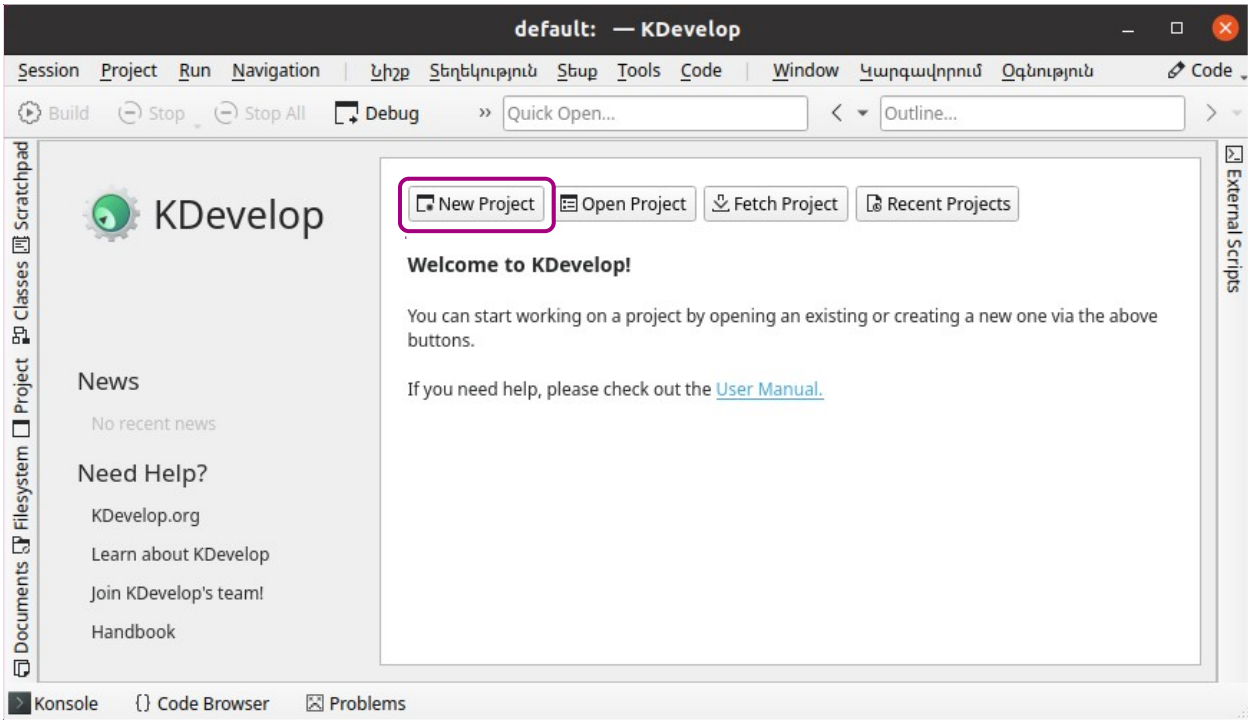


Նկար 4

1.5. Գրենք առաջին C ծրագիրը

Ցանկացած ծրագրավորման լեզու սովորելիս՝ 1-ինը գրում են պարզագույն ծրագիր, որը կարտածի «**Hello World**» տեքստը: Մեր դեպքում օգտագործենք «Բարև աշխարհ» հայկական տարբերակը: Ծրագիրը հավաքենք kdevelop միջավայրում, իսկ մինչ այդ ստեղծենք նախագիծ՝ «ԲարևԱշխարհ» անունով:

New Project



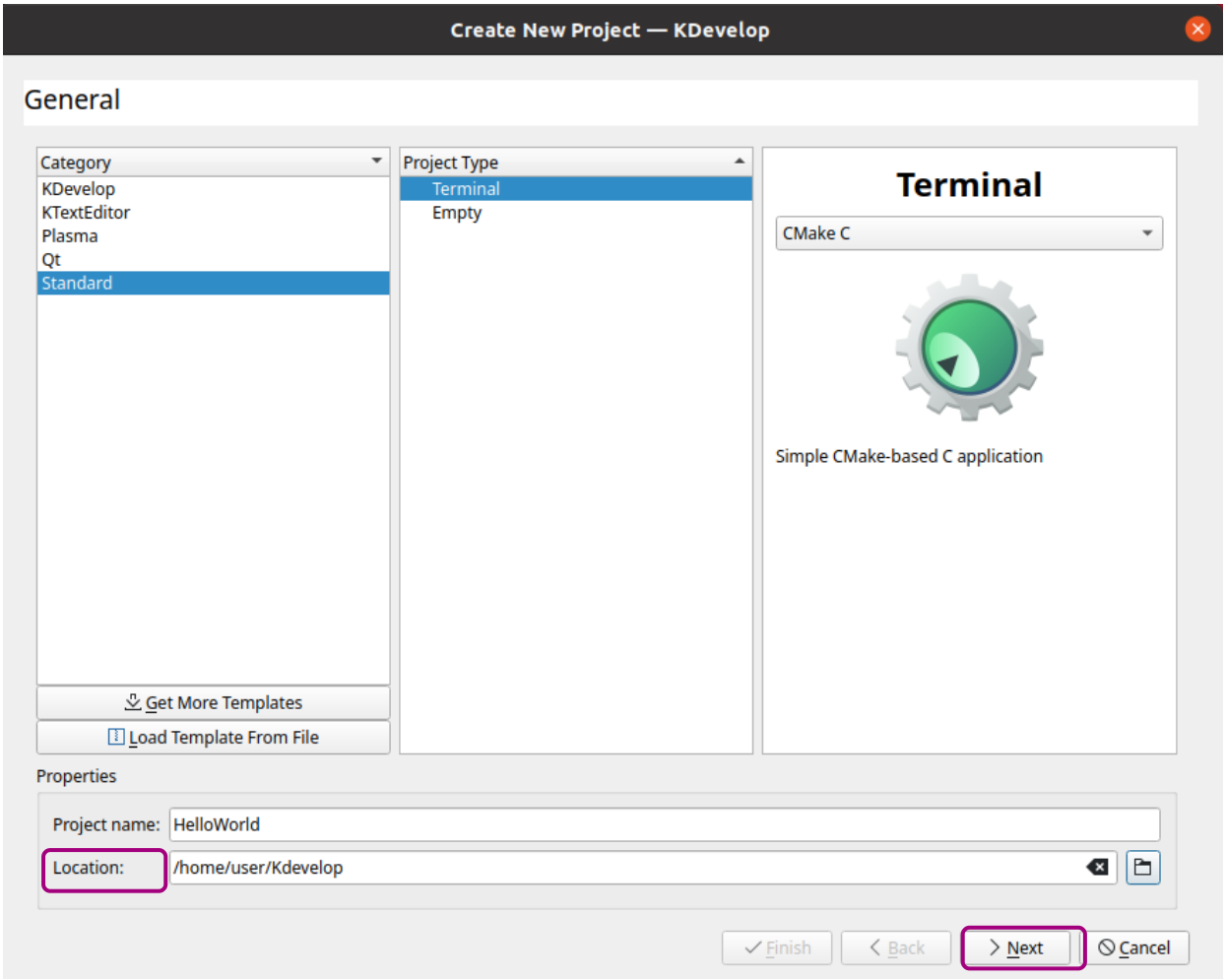
Նկար 5

ProjectName – նախագծի անունը լատինատառ

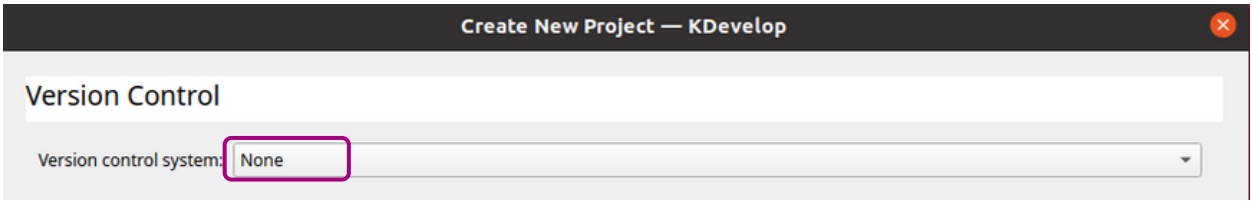
Category – Standart

Project Type – Terminal

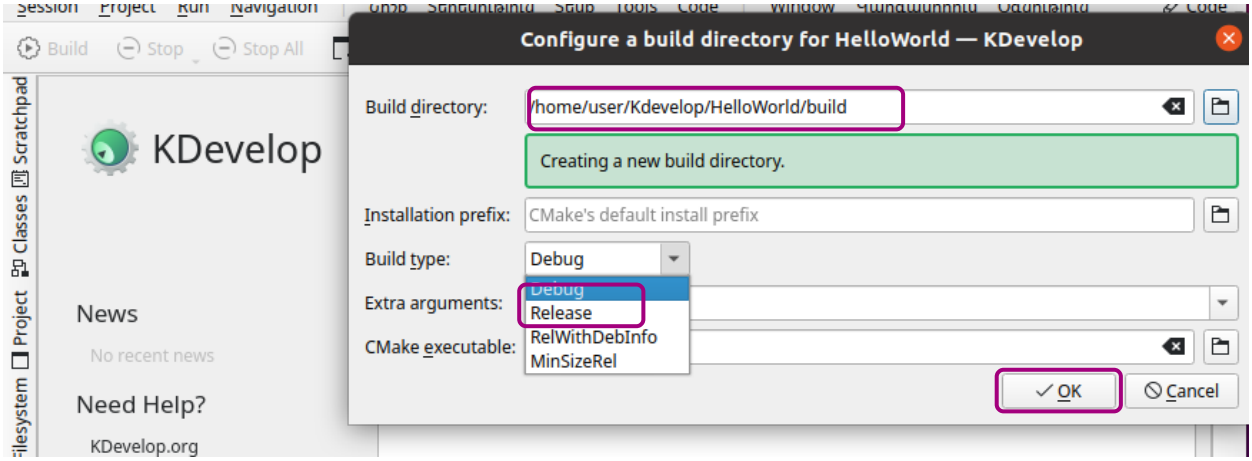
Location - նախագծի պանակը



Նկար 6

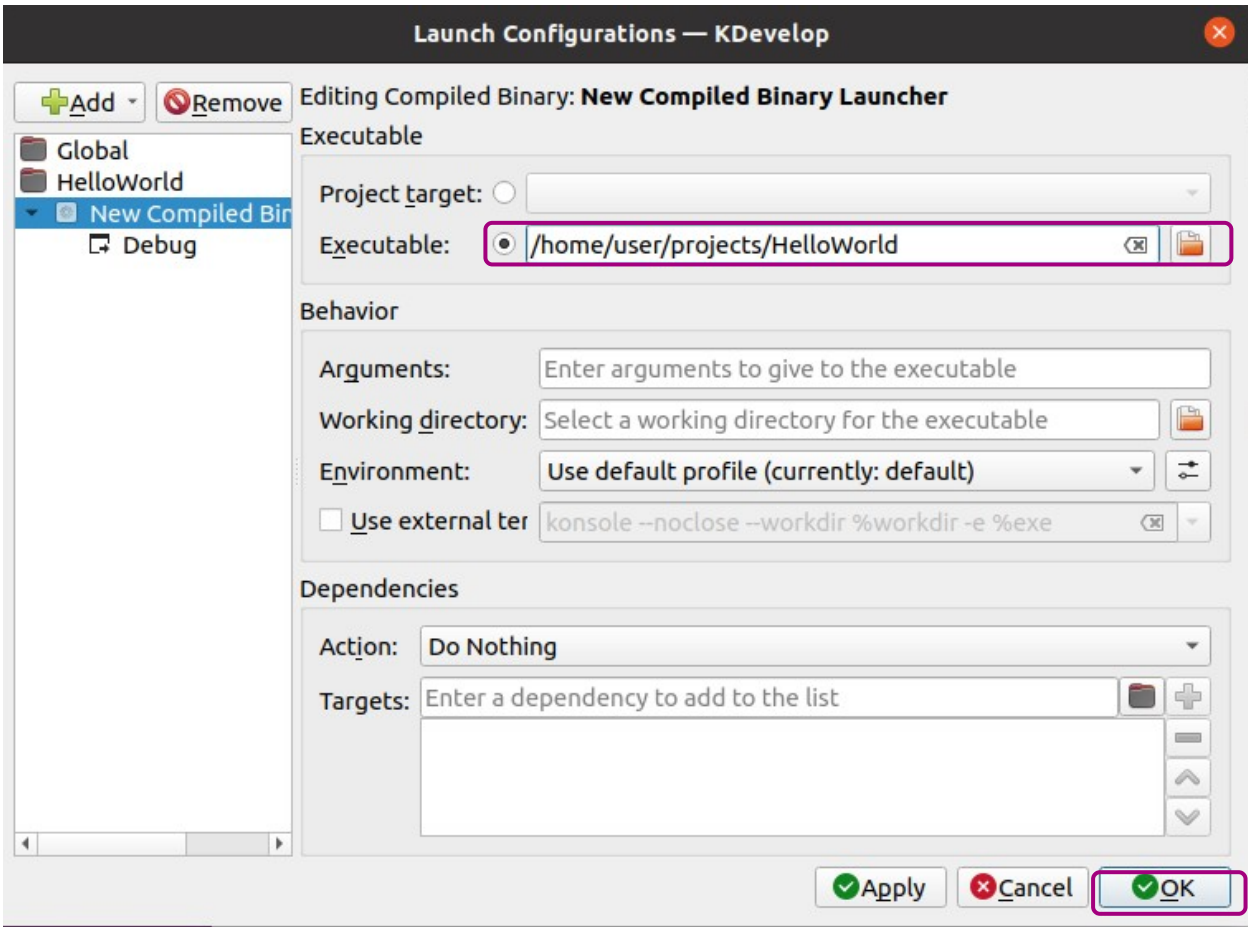


Նկար 7

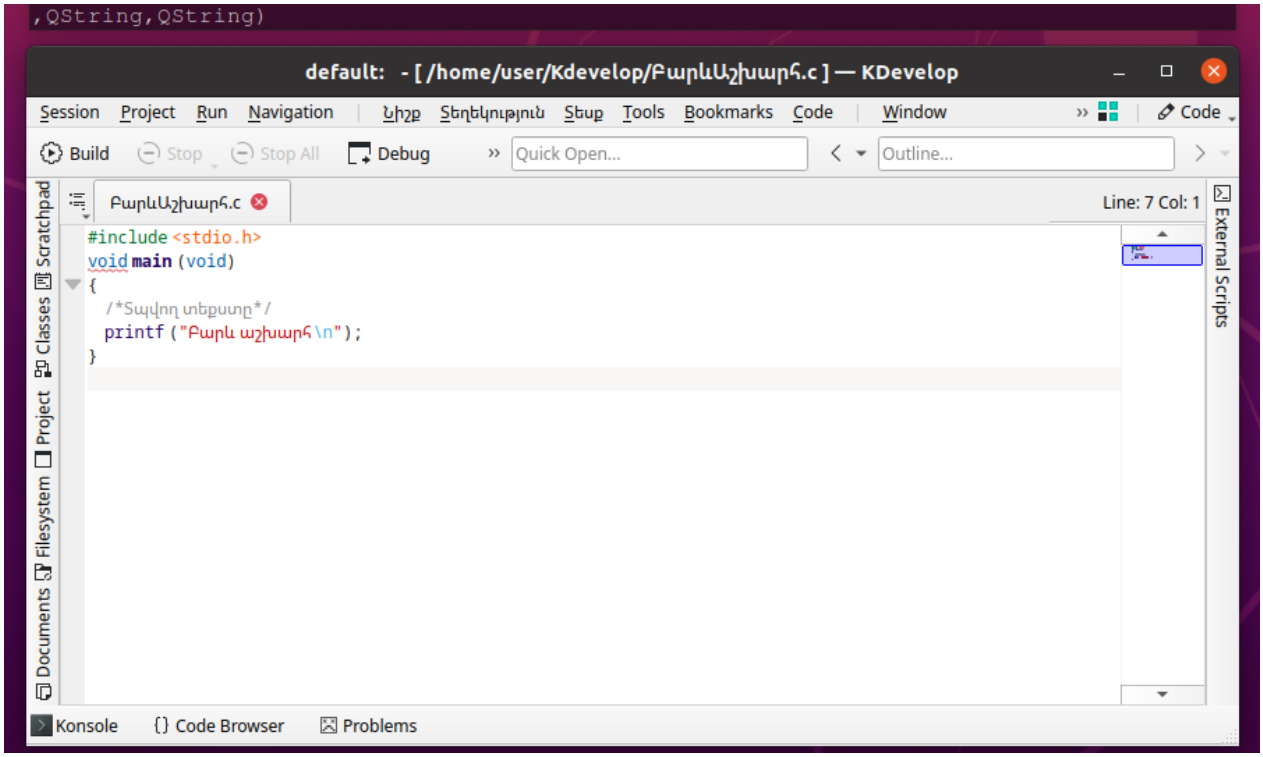


Նկար 8

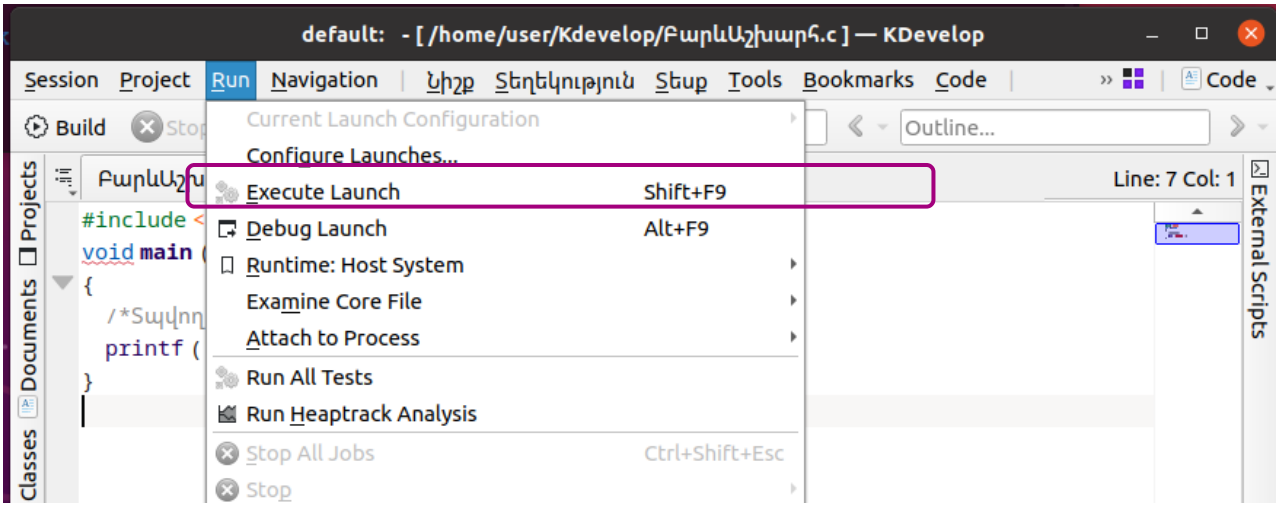
Build և Run | Կառուցել և գործարկել



Նկար 9

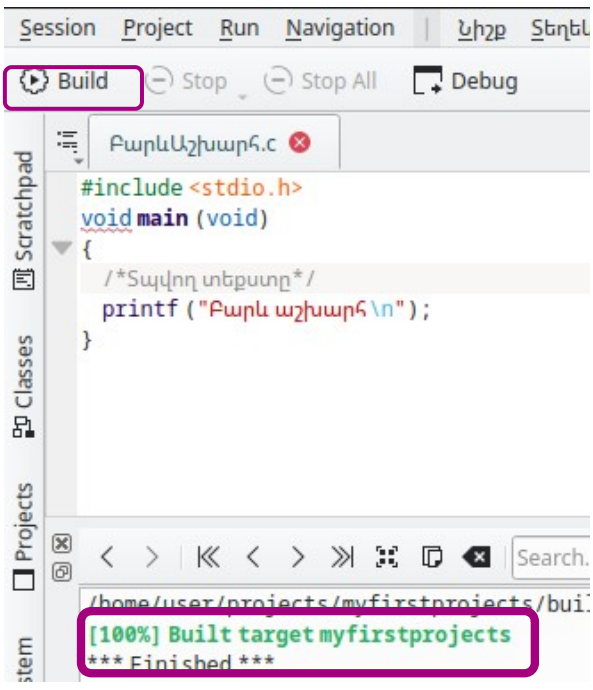


Նկար 10

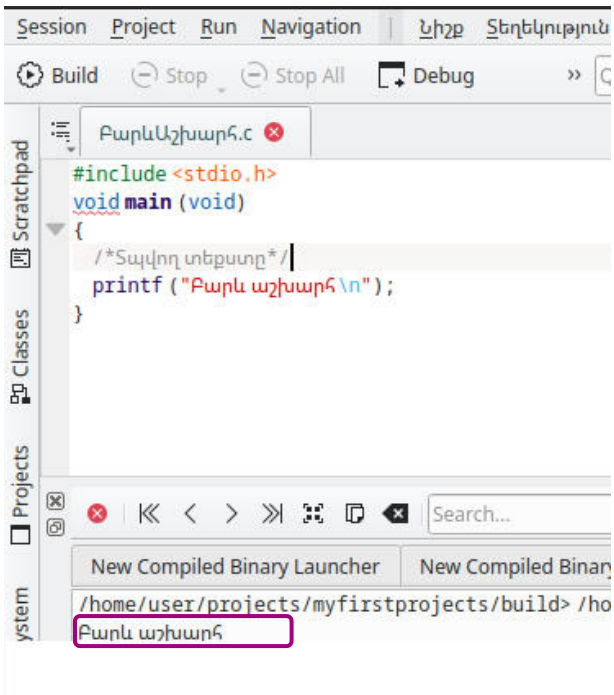


Նկար 11

Կարգաբերումները կատարելուց հետո կարող ենք գործարկել ծրագիրը ՝ Նախ սեղմենք Build - կառուցել կոճակը, հետո Run/Execute Launches կամ Shift+F9:

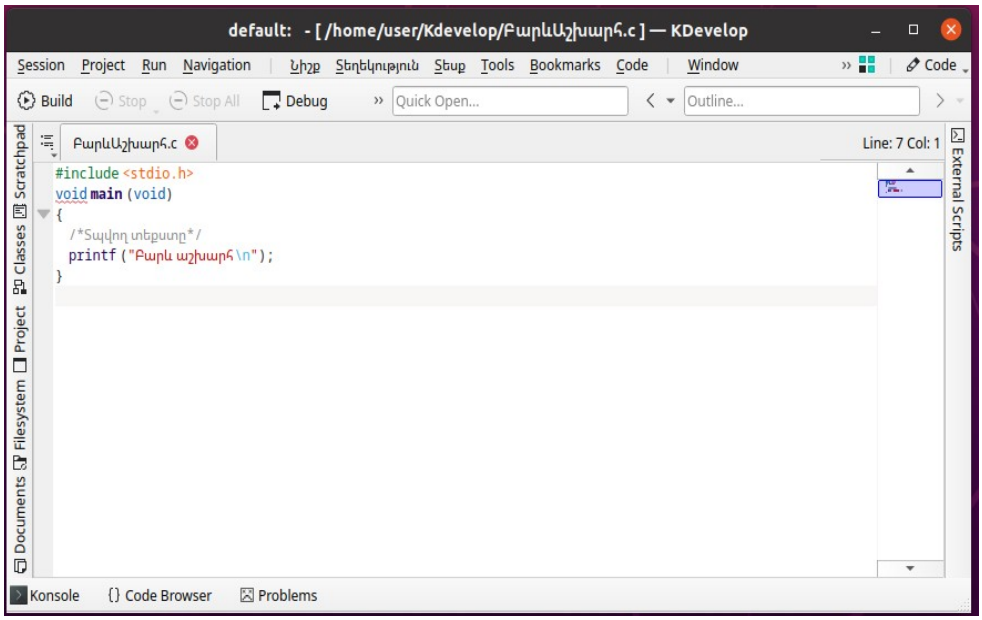


Նկար 12



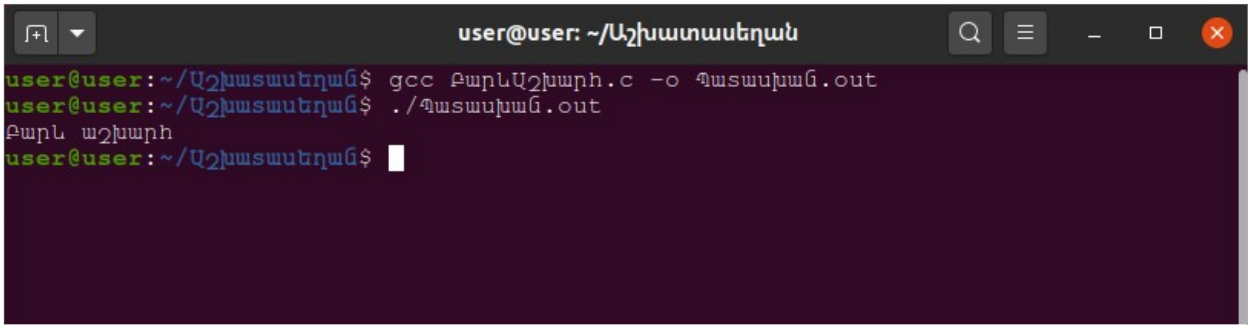
Նկար 13

Ցանկացած տեսակի գրաֆիկական միջերես ունեցող խմբագրիչով «IDE» գործարկելու դեպքում մոտավոր նույնությամբ վերոնշյալ գործողությունները պետք է կատարել։ Սակայն, եթե օգտագործում ենք terminal, ապա ծրագիրը կոմպիլացնելու համար ընդամենը 4 պարզ հրահանգ է անհրաժեշտ.



Նկար 14

- sudo nano ԲարևԱշխարհ.c (nano-ն տերմինալային խմբագրիչ է)
- խմբագրում ենք կոդը (#include <stdio.h>...), Ctr+X, y
- gcc ԲարևԱշխարհ.c -o Պատասխան.out (-o ստղեծում է գործարկվող նիշք)
- ./Պատասխան.out



Նկար 15

Եկեք հերթով ուսումնասիրենք ծրագրի տողերը և սովորենք C-ի պարզ այլագրերը:

- **#include <stdio.h>**
  hash-include - «հեշ» ներառումներ, որի միջոցով ծրագիրը ներառում է մի շարք ֆունկցիաների գրադարաններ (printf, stdio և այլն): **stdio**-ն դա standard input/output-ի կրճատ տարբերակն է՝

ստանդարտ ներմուծում/արտածում: stdio**.h (header)** – այն նույնպես C լեզվի նիշք է, սակայն .h վերջավորությամբ: Այն պարունակում է հայտարարագրեր և մակրո սահմանաումներ, որոնք կարող են համօգտագործվել տարբեր աղբյուրներում: Առհասարակ C ծրագրում կարելի է ներառել հետևյալ .h տեսակի ֆունկցիաները՝

<assert.h>-Diagnostics Functions – Ախտորոշող ֆունկցիաներ
<ctype.h>-Character Handling Functions - Նիշերի մշակման ֆունկցիաներ
<locale.h>-Localization Functions – Տեղայնացման/լեզվի ֆունկցիաներ
<math.h>-Mathematics Functions – Մաթեմատիկական ֆունկցիաներ
<setjmp.h>-Nonlocal Jump Functions – Ոչ տեղային ցատկող ֆունկցիաներ
<signal.h>-Signal Handling Functions – Ազդանշանի մշակման ֆունկցիա
<stdarg.h>-Variable Argument List Functions – Փոփոխ., արգ. ցուցակի ֆունկցիա
<stdio.h>-Input/Output Functions – ներմուծման/արտածման ֆունկցիա
<stdlib.h>-General Utility Functions – Հիմնական պիտույքի ֆունկցիա
<string.h>-String Functions – Տողային ֆունկցիա
<time.h>-Date and Time Functions – Ամսաթվի և ժամանակի ֆունկցիա

- **void main (void)**
  C-ն ֆունկցիաների վրա հիմնված լեզու է և ցանկացած նախագիծ բաղկացած է մի շարք ֆունկցիաներից: Յուրաքանչյուր ֆունկցիա վերցնում է 0 կամ ավել արգումենտներ (փոփոխական) և հետ է վերադարձնում մեկ արժեք: C-ում ֆունկցիան ունի սահմանումներ, և այս դեպքում այն արտահայտվում է հետևյալ կերպ՝
  - void – այն, ինչ ֆունկցիան հետ է վերադարձնում,
  - main – ֆունկցիայի անունը,
  - () արգումենտների ցանկը փակագծերի մեջ, այս դեպքում՝ «void»:
  Ցանկացած C ծրագիր պետք է պարունակի main ֆունկցիա, քանի որ երբ ծրագիրը կոմպիլացնում ենք, առաջինը հենց main-ն է գործարկվում:
  Void նշանակում է դատարկ, ճշտում կամ նշում է տեսակը և չի պահանջում արժեք: Բացի void-ից կան մի շարք այլ տեսակներ:
- /* Տպվող տեքստը */
  /* ... */ C ծրագրավորման մեջ մեկնաբանությունները գրվում են հենց այդպես՝ «թեք գիծ աստղանիշ .... աստղանիշ թեք գիծ»: Մեկնաբանության մեջ գրված տեքստը չի կոմպիլացվում:
  (/ = slash = թեք գիծ), (\ = backslash = հակադիր թեք գիծ), (* = asterisk = աստղանիշ)
- **printf ("Hello world!\n");**
  printf ֆունկցիան կանչ է կատարում stdio գրադարանից (Print formatted – ձևաչափված արտածում):
  ; - C ծրագրավորման լեզվի մեջ բոլոր տեսակի արտածումների տողերի վերջում (տպել – print) պարտադիր է կետ-ստորակետը:
  \n – կոչվում է escape sequence, այն նոր տողի համար է:

1.6. Թվաբանական օպերատորներ

C ծրագրավորման լեզուն ունի մի շարք թվաբանական օպերատորներ, ուսումնասիրենք դրանցից մի քանիսը և ներկայացնենք տեսակները։

Նախ ներկայացնենք տվյալների տիպերը, որոնք ամենաշատն են օգտագործվում։

Ամբողջ թվեր

Int – integers

C- ում ամենից շատ օգտագործվում են «ամբողջ թիվ» տիպի տվյալներ։

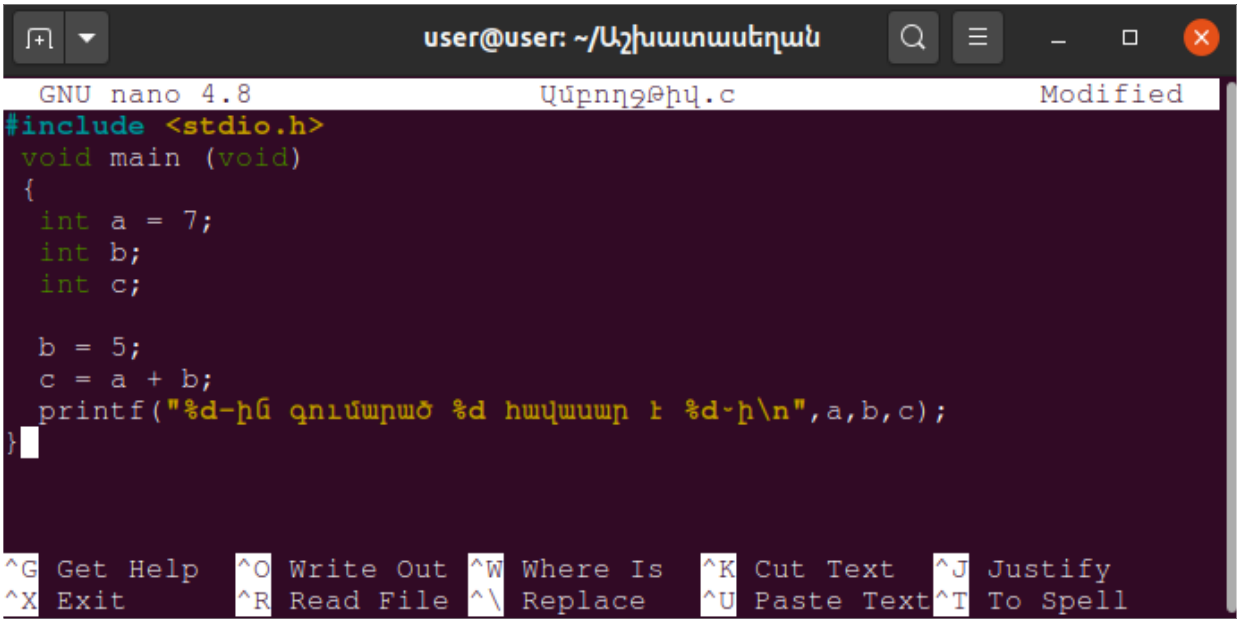
```c
#include <stdio.h>
 void main (void)
 {
  int a = 7;
  int b;
  int c;

  b = 5;
  c = a + b;
  printf ("%d-ին գումարած %d հավասար է %d-ի\n", a, b, c);
}
```

Ծրագիրը հավաքենք տերմինալում `

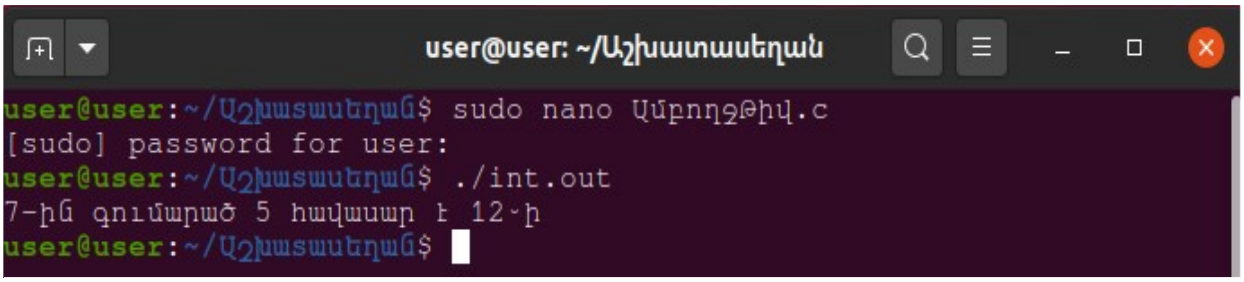
touch Ամբողջթիվ.c - ստեղծենք նիշքը,

sudo nano Ամբողջթիվ.c - հավաքենք կոդը nano խմբագրիչով։



Նկար 16

Եթե ցանկանում եք տերմինալը աշխատացնեք windows ՕՀ-ից, ապա կարող եք ակտիվացնել կամ ներբեռնել WSL ենթածրագիրը ՝ Ubuntu terminal:



Նկար 17

```
int a = 7;
int b;
int c;
```

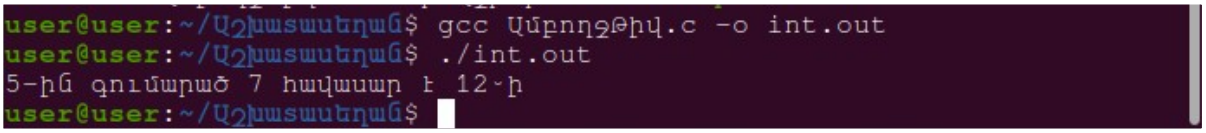
Այս երեք տողերը հանդիսանում են main ֆունկցիայի հայտարարագրերը, որոնք կոմպիլատորին թելադրում են, որ պետք է օգտագործել a, b, c փոփոխականները, որոնք ամբողջ թվեր են՝ int։
Կարող ենք կատարել նաև մի քանի հայտարարագրումներ մեկ տողում, ստորակետներով բաժանված՝ **int a=7, b, c;** որն ավելի կրճատ ձև է։
Ինչպես տեսնում եք, 1-ին տողում բացի փոփոխական հայտարարագրելուց, a-ին նաև վերագրել ենք 7 արժեք, որը կարող ենք կոչել սկզբնարժեք։ Իսկ b և c փոփոխականների արժեքը սահմանված չէ(նշենք, որ այն 0 չէ)։

```
b = 5;
c = a + b;
```

Հաջորդ 2 տողում կատարում ենք մաթեմատիկական գործողություն։

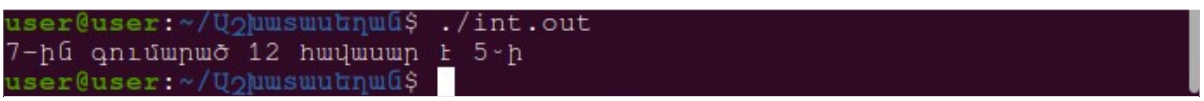
```
printf ("%d-ին գումարած %d հավասար է %d-ի\n", a, b, c);
```

Վերջին տողը հենց բուն գործողության արտածման/տպման print ֆունցկիան է։ Յուրաքանչյուր %d-ն փոխարինում է a, b, c -ի արժեքները տասնորդական տեսքով (d = decimal = տասնորդական)։ Ստորակետից հետո նշվող a, b, c -ով սահմանում ենք %d-ի հերթականությունը։
Օրինակ, եթե գրենք b, a, c, ապա ծրագիրը կաշխատի հետևյալ կերպ՝



Նկար 18

Իսկ a, c, b -ի դեպքում՝



Նկար 19

Նկար 19

Ինչը իհարկե սխալ հաշվարկ է, սակայն հիշենք, որ մենք մեր ծրագիրը գրել ենք այնպես, որ printf ֆունկցիան տպի “... “-ում եղածը որպես նիշ(symbol), այլ ոչ թե որպես մաթեմատիկական գործողություն։

Կոտորակներ կամ տասնորդական թվեր

Կոտորակները կամ տասնորդական թվերը ($\frac{5}{6}$; 0.1; 0.7; 0.9 և այլն) հնարավորություն են տալիս գրել ոչ ամբողջ թվեր։ Կոտորակներով կարելի է լրացնել ամբողջ թվերի միջև եղած միջակայքերը, իսկ դա հնարավորություն է տալիս ավելի ճշգրիտ չափումներ կատարել։

Այս դեպքում int-ի փոխարեն օգտագործում ենք float տերմինը։

float a;

float-ը կոմպիլատորին թելադրում է, որ a-ն ոչ-ամբողջ թիվ է։ Փորձենք նախորդ ծրագիրը float տարբերակով։ Պատճենենք «ԱմբողջԹվեր.c» նիշքը և վերանվանենք որպես «Կոտորակ.c»՝

sudo cp ԱմբողջԹվեր.c Կոտորակ.c

```
user@user:~/Աշխատասեղան$ sudo cp ԱմբողջԹիվ.c Կոտորակ.c
[sudo] password for user:
user@user:~/Աշխատասեղան$ ls
int.out  ԱմբողջԹիվ.c  ՔարևԱշխարհ.c  Կոտորակ.c  Պատասխան.out
user@user:~/Աշխատասեղան$
```

Նկար 20

Նկար 21

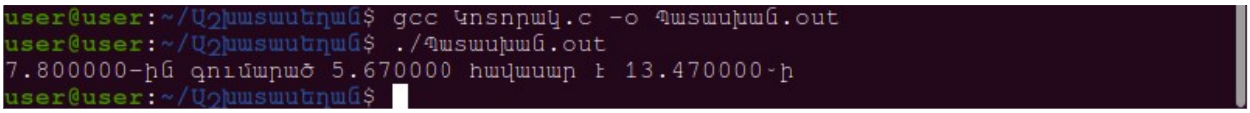
Կոմպիլացնենք ծրագիրը՝ gcc Կոտորակ.c o Պատասխան.out։

Կոմպիլացման ընթացքը և ծրագրի պատասխանը կարող եք տեսնել նկար 22-ում։ Տեսնում ենք, որ բացասական փոքր թվեր են, որոնք կապ չունեն առկա փոփոխականների հետ։ Այստեղ խնդիրը հենց %d-ն է, որը կոտորակային թվերի դեպքում պետք է փոխարինել %f-ով։

```
Կոտորակ.c:10:58: warning: format ‘%d’ expects argument of type ‘int’, but argument 4 has type ‘double’ [-Wformat=]
   10 |   printf("%d-ին գումարած %d հավասար է %d-ի\n",a, c, b);
      |                                                ~^       ~
      |                                                 |       |
      |                                                 int     double
      |                                                %f
user@user:~/Աշխատասեղան$ ./Պատասխան.out
-325518648-ին գումարած -325518632 հավասար է -1273613904-ի
user@user:~/Աշխատասեղան$
```

Նկար 22

Փոխենք թվերը ՝ a=7.8, b=56.57 և կրկին փորձենք ծրագիրը։



Նկար 23

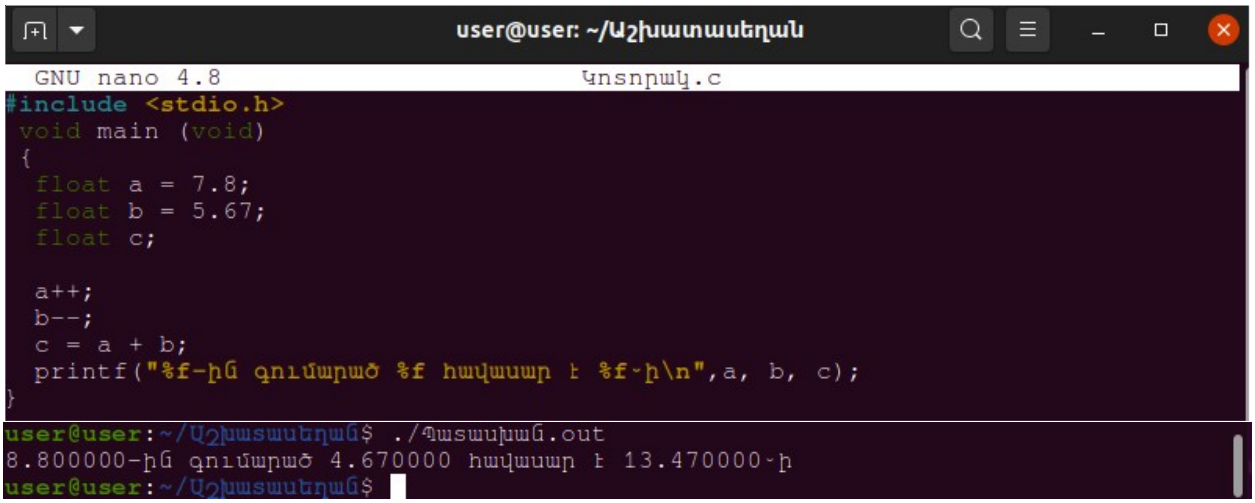
Ի դեպ, փոփոխականները պետք է լինեն լատինատառ, կարող եք փորձել, օրինակ, հայերեն կամ ռուսերեն տառերով և կտեսնեք, որ ծրագիրը չի կոմպիլացվում (compilation = կազմաձևում, փոխակերպում է մեքենային հասկանալի ցածր լեզվի ՝ 0 կամ 1)։

C լեզուն ունի նաև հակիրճ գրելաձև տարբեր մաթեմատիկական գործողությունների համար ՝

- Գումարում և հանում

  a = a + 1 <=> a++

  b = b + 1 <=> b--



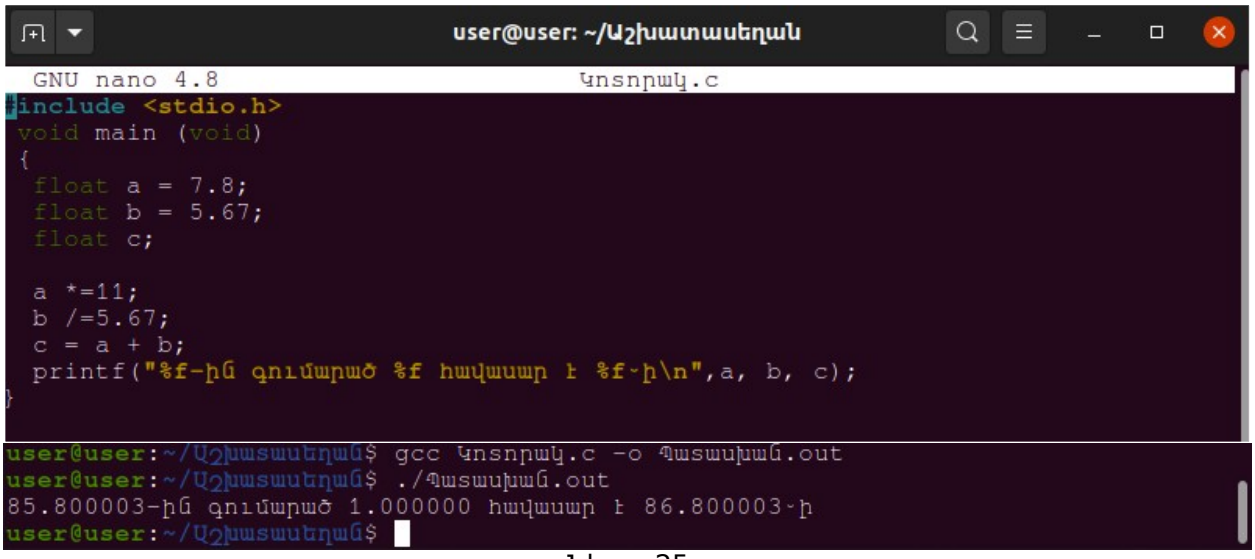
Նկար 24

- Բազմապատկում և բաժանում

  a = a *11 <=> a *= 11

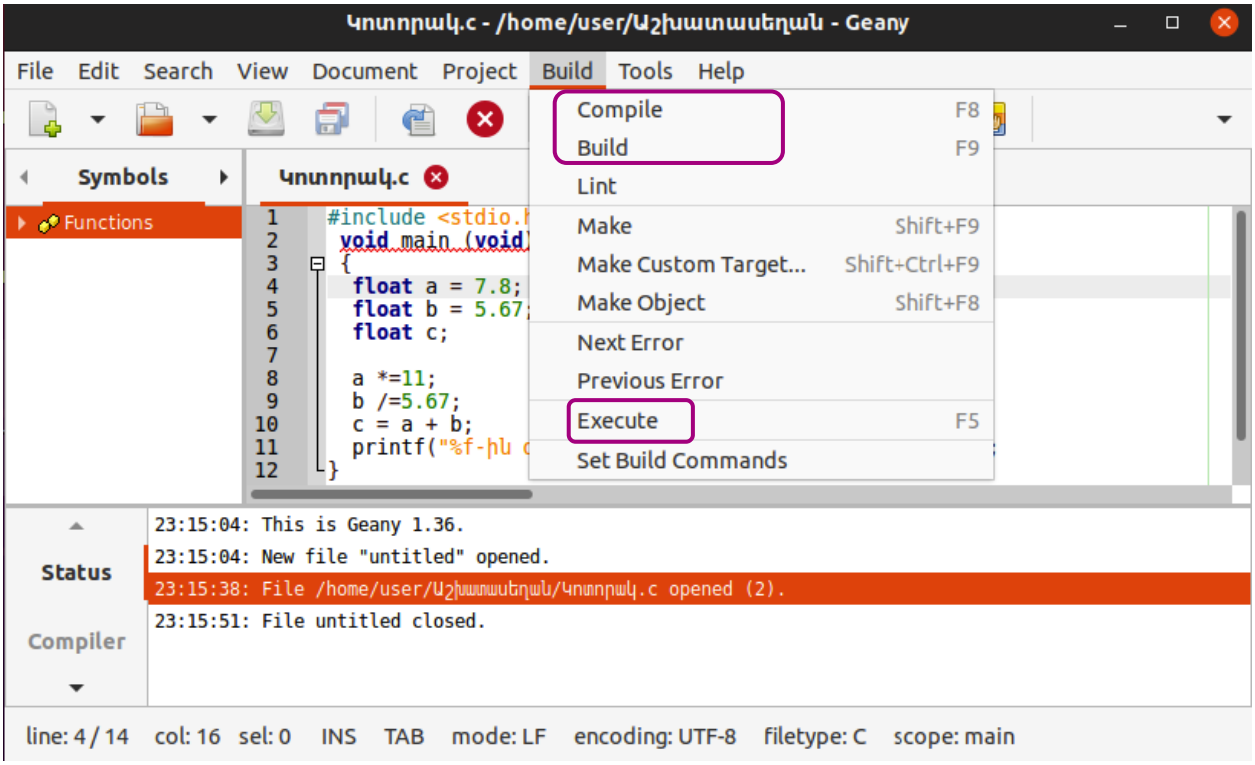
  b = b / 5.67 <=> b /=5.67

Եթե ցանկանում եք ունենալ Linux/Debian/Ubuntu ՕՀ-ն, առանց ջնջելու Ձեր windows ՕՀ-ն, ապա կարող եք տեղադրել VirtualBox ծրագիրը և VirtualBox-ի մեջ առանձին տեղծեք նոր ՕՀ-ն, որը ոչնչով չի խանգարելու Windows-ի ՕՀ-ին:



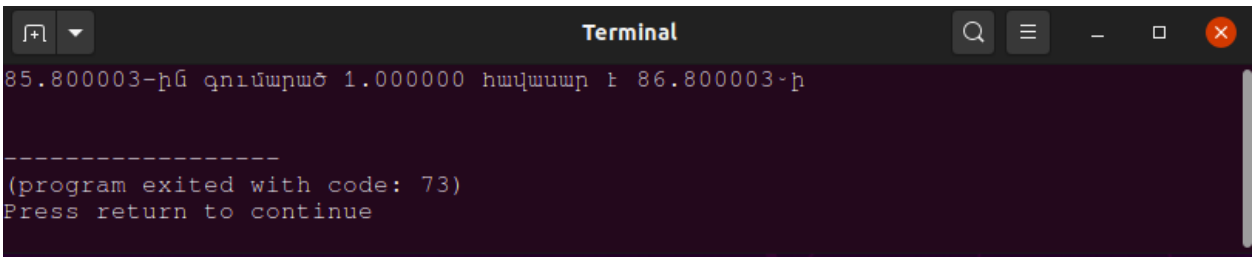
Նկար 25

Եթե շատ եք ձանձրանում տերմնալային խմբագրիչից, ապա կարող եք օգտագործել ավելի պարզ գործիք, որը կոչվում է **Geany**՝



Նկար 26

Compile -կազմաձևել- **F8**, Build –կառուցել- **F9**, Execute -գործարկել - **F5**



Նկար 27

%f գրելով՝ ծրագիրը տպում է կոտորակային թվի կետից հետո ևս 6 թիվ, այսինքն հարյուրհազարերորդական մասը: Իհարկե որքան մեծ, այնքան ավելի ճշգրիտ կլինի արդյունքը, սակայն կարող ենք այն փոխել և ծրագիրը կտպի .-ից հետո n թիվ՝ օգտագործելով %.nf ձևաչափը, որտեղ n-ը կետից հետո գրվող n-երրորդական մասն է: Ընդունեք՝ n=2, n=10 կամ օգտագործենք տարբեր թվեր և ստուգենք ծրագիրը՝

```
  printf("%.2f-ին գումարած %.2f հավասար է %.2f-ի\n",a, b, c);
}
user@user:~/Աշխատանք$ gcc կոտորակ.c -o Պատախան.out
user@user:~/Աշխատանք$ ./Պատախան.out
85.80-ին գումարած 1.00 հավասար է 86.80-ի
```

```
  printf("%.10f-ին գումարած %.10f հավասար է %.10f-ի\n",a, b, c);
}
user@user:~/Աշխատանք$ gcc կոտորակ.c -o Պատախան.out
user@user:~/Աշխատանք$ ./Պատախան.out
85.8000030518-ին գումարած 1.0000000000 հավասար է 86.8000030518-ի
user@user:~/Աշխատանք$
```

```
  printf("%.10f-ին գումարած %.25f հավասար է %.100f-ի\n",a, b, c);
}
user@user:~/Աշխատանք$ gcc կոտորակ.c -o Պատախան.out
user@user:~/Աշխատանք$ ./Պատախան.out
85.8000030518-ին գումարած 1.0000000000000000000000000 հավասար է 86.8000030517578125000000000000000000000000000000000000000000000000000000000000000000000000000000000000000-ի
```

Նկար 28

Նիշային փոփոխականներ

Char/նիշ տիպի փոփոխականները ևս ամենաօգտագործվող տիպերից է C ծրագրավորման մեջ: Սա օգտագործում են այն ժամանակ, երբ ցանկանում են պահել որպես նիշ (տառեր, թվեր և կետադրական նշաններ): Գոյություն ունեն նիշային կոդավորման մի քանի ստանդարտ համակարգեր՝ ASCII, UTF-8, UTF-16, UTF-32 և այլն: Սրանցից ամեն մեկն ունի իր սեփական թվային համակարգը, օրինակ՝ ASCII-ն օգտագործում է 0-ից մինչև 127 արժեք և ցանկացած նիշի համար զբաղեցնում է 1 բայթ ծավալ: Char-ում և Int-ում կան համընկնում՝ կապված դրական և բացասական թվերի հետ, սակայն, եթե ուզում ենք աշխատել միայն դրական թվերի հետ, ապա պետք է օգտագործենք unsigned փոխարկիչը:

char a; հայտարարագրում է a նիշ, որը կարող է պահել -128-ից 127 միջակայքով արժեքներ:

unsigned char b; հայտարարագրում է b նիշ, որը կարող է պահել 0-ից 255 միջակայքով արժեքներ:

Նիշերով թվաբանական գործողություններ անելը սխալ է, քանի որ այն կարող է լինի մինչև 127 թվային արժեք և դրանից ավելին համակարգում կլինի գործողությունների գերհոսք և կունենանք ծրագրային վրիպակներ (bugs): Բայց եթե խնդրի դրվածքն այնպես է, որ պետք է օգտագործենք թվաբանական խնդիր լուծելու համար, ապա հնարավոր է կատարել, միայն թե վերջնական արդյունքը չպետք է հատի 127 թիվը:

1.7. Համեմատական օպերատորներ, պայմաններ

Ծրագրավորման մեջ ամենակարևոր հիմնային կետերից են պայմանները, քանի որ դրանցով կարող ենք կատարել տարբեր գործողություններ, ճկունությամբ կառավարել ծրագիրը և հեշտորեն իրագործել խնդրի պահանջը։

եթե, այլապես | if – else

Տերմինալում ստեղծենք ԵթեԱյլապես.c նիշքը և հավաքենք հետևյալ սցենարը՝

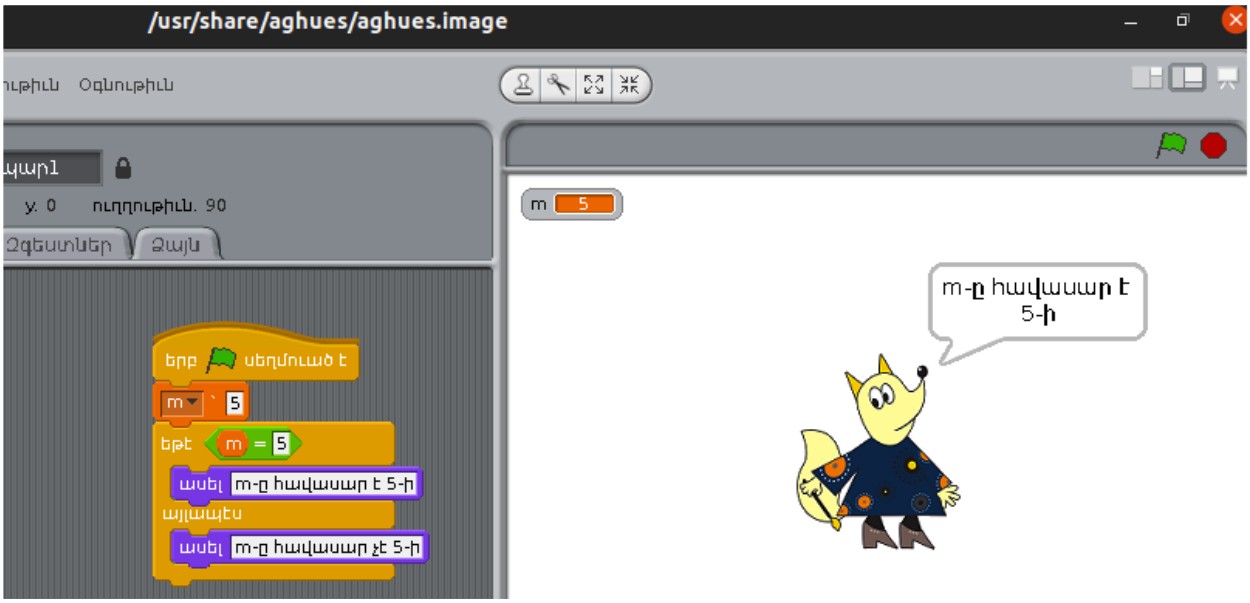
```
#include <stdio.h>
 void main (void)
 {
    int m = 5;

    if (m == 5) {
      printf ("a-ն հավասար է 5-ի\n"); // լուծում-1
    }
    else
    {
       printf ("a-ն հավասար չէ 5-ի\n"); // լուծում-2
    }
 }
```

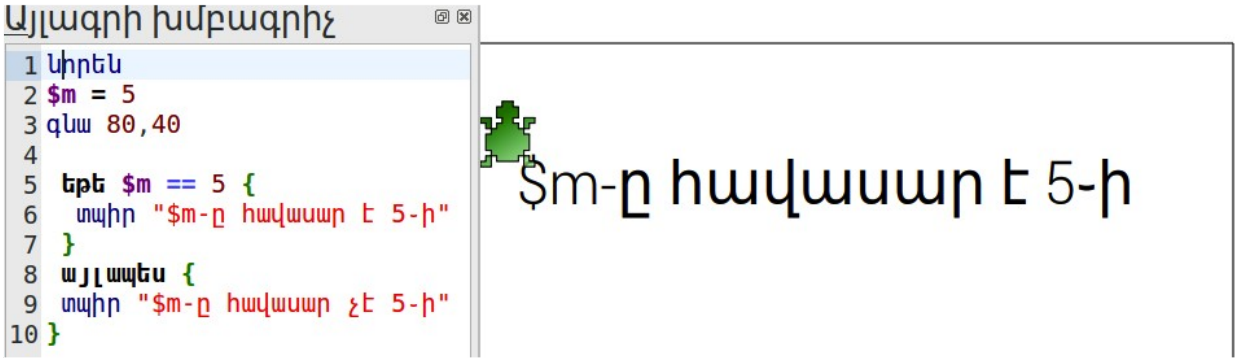
int m = 5; հայտարարագրում ենք m-ը, որպես ամբողջ տիպի փոփոխական և տալիս ենք 5 սկզբնարժեք։ if (m==5)-ով պայման ենք դնում, եթե m-ը հավասար է 5-ի, ապա տպել լուծում 1-ը, հակառակ դեպքում կամ այլապես տպել լուծում 2-րդը։

Նկար 29

Եկեք այս ծրագիրը համեմատենք Աղուէս և Կրիայ լեզուներով գրված համանման սցենարների հետ՝



Նկար 30



Նկար 31

Ինչպես տեսնում ենք Աղուէս, Կրիայ-ով խնդիրը ևս լուծելի է, իսկ այլագրերը շատ նման են։

Ինչպես C-ում, այնպես էլ Կրիայում = նշանակում է, որ m-ին հստակ վերագրում ենք արժեք, իսկ == փորձարկում ենք m-ին վերագրվող արժեքը։

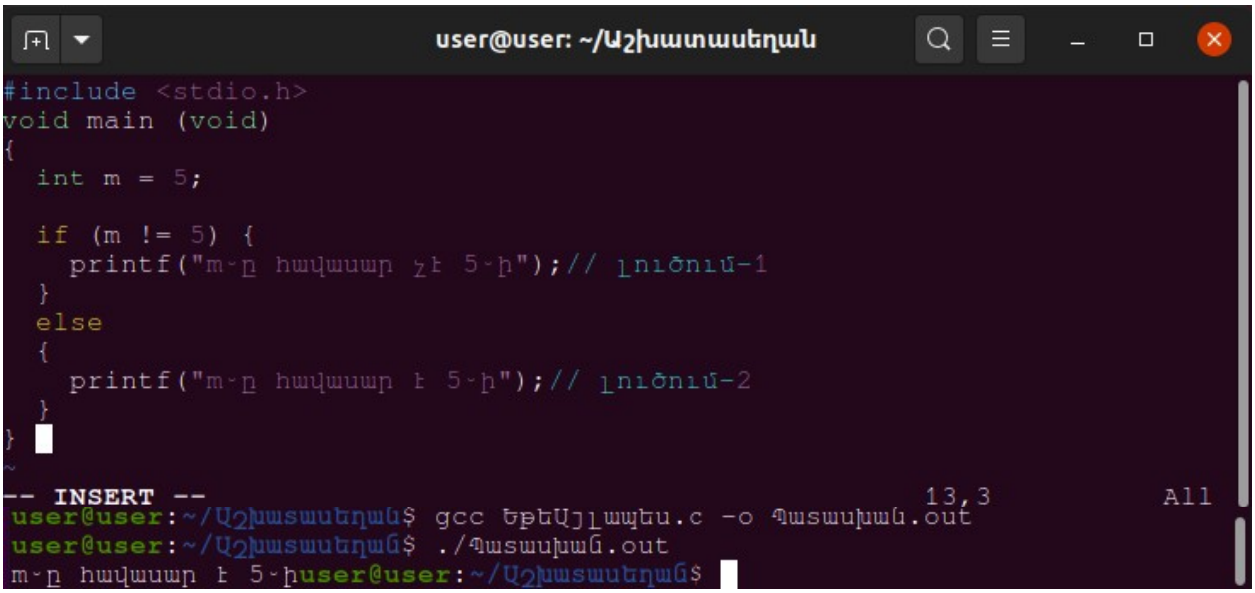
Բացի մեկ = և երկու == հավասարից կարող է լինել նաև != գրելաձև, որը նշանակում է հավասար չէ։ Եկեք գրենք համանման սցենար != ի համար և այս դեպքում էլ օգտագործենք մի նոր խմբագրիչ, որը կոչվում է Vim։ Vim խմբագրիչը որոշ ՕՀ-երում լռելյայն տեղադրված է լինում, սակայն, եթե չունեք, ապա կարո եք տեղադրել՝ հավաքելով հետևյալ հրամանը ՝ Debian/Ubuntu – sudo apt install vim կամ snap install vim ։

Բացենք ԵթեԱյլապես.c նիշքը vim խմբագրիչով ՝ sudo vim ԵթեԱյլապես.c ։

Սեղմեք մեծատառ A տառը և սկսեք խմբագրել (ծանոթանալ vim հրամաններին)։

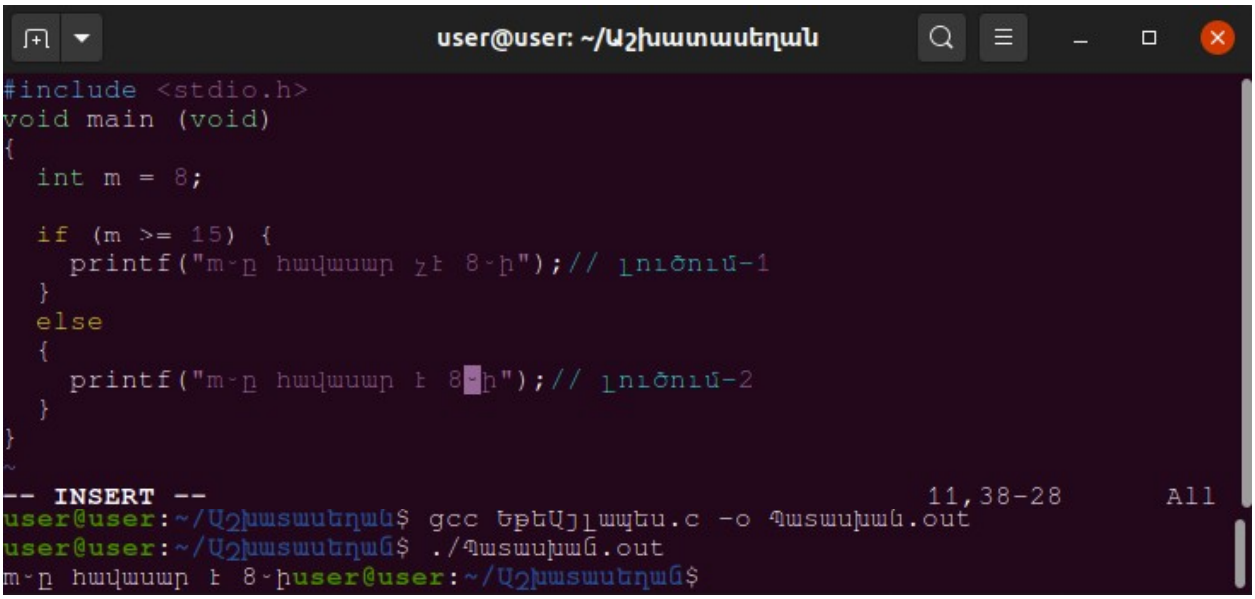
Այն նման է GNU nano խմբագրիչին, սակայն ունի ավելի շատ հնարավորություններ և արագ կոդ գրելու համար կարճ ստեղներ։

Խմբագրումը կատարելուց հետո գրեք Alt: , որի միջոցով դուրս եք գալիս կուրսորի գրառման ռեժիմից և :w ու enter (պահում է գրառումը / write), ապա փակել vim խմբագրիչը՝ :q! և enter (quit)։



Նկար 32

Բացի = ից կարող ենք օգտագործել նաև մեծ > ու <փոքր, մեծ կամ հավասար >= և <= փոքր կամ հավասար օպերատորները։



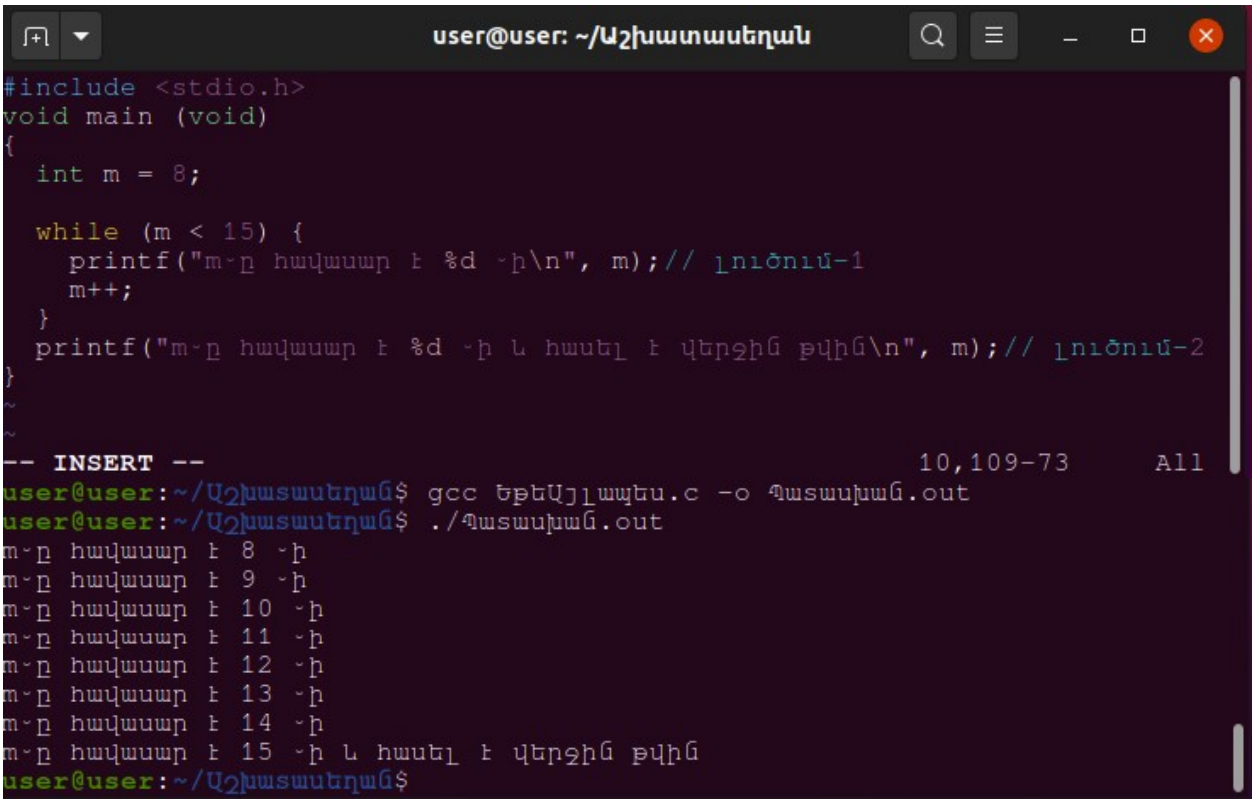
Նկար 33

Ցիկլեր | Loops

While - Քանի դեռ

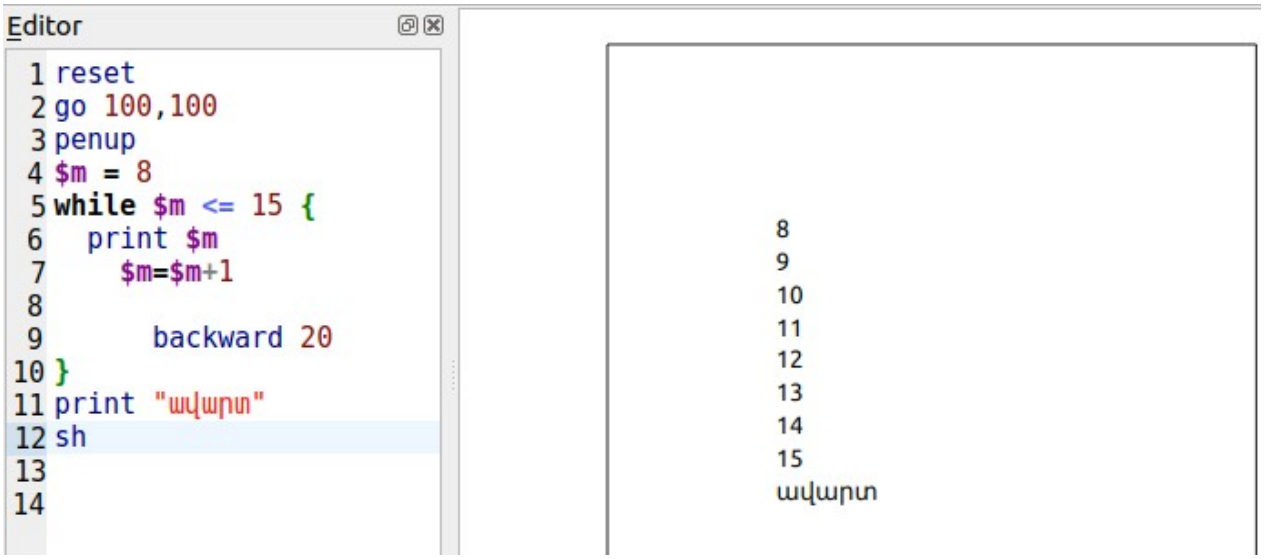
Ցիկլերը ևս դասվում են պայմանական և համեմատական օպերատորների շարքին։ While / քանի դեռ ցիկլը շատ նման է if

պայմանին, եթե ուզում եք օգտագործել պայմանը այնքան ժամանակ քանի դեռ կամ **մինչ** այն կլինի ճիշտ կամ սխալ (true/false)։



Նկար 33

Եկեք այս խնդիրը լուծենք նաև Kturtle-ով(օգտագործեք Կրիայ, եթե ցանկանում եք գրեք հայերեն այլագրով)։



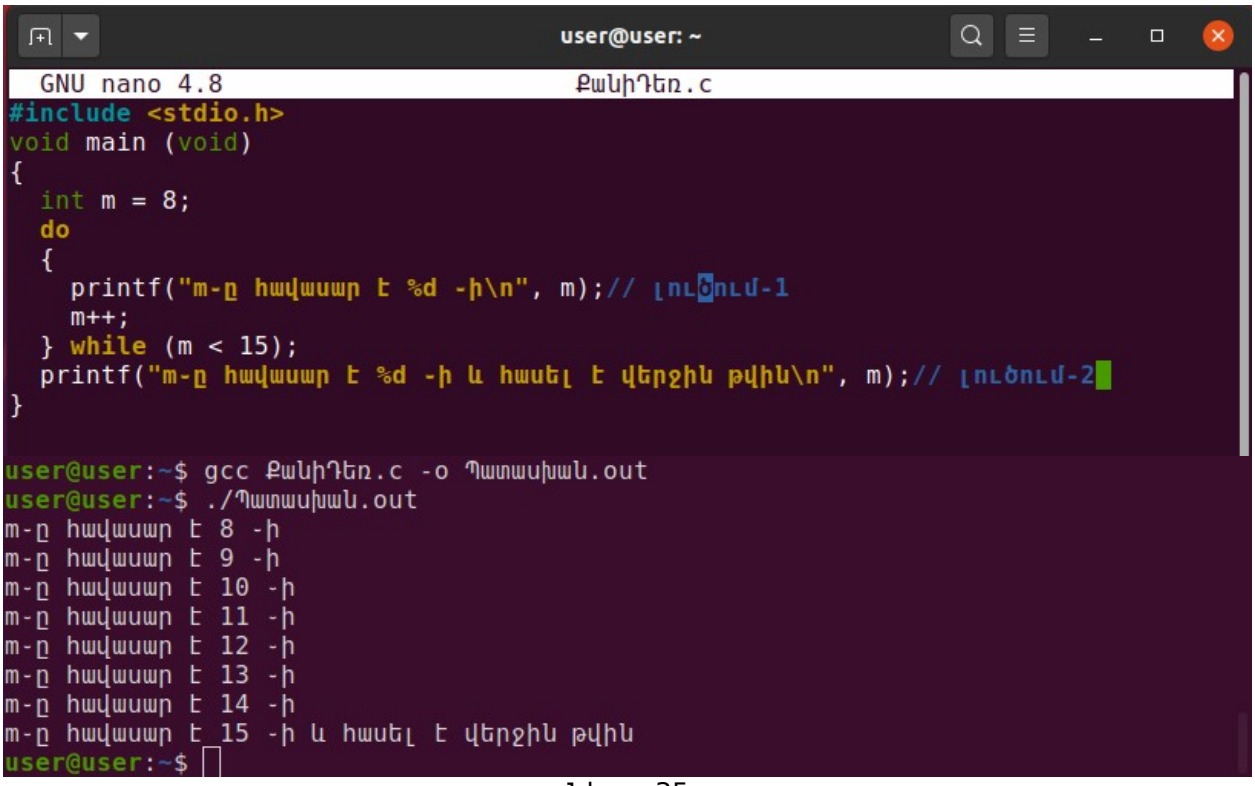
Նկար 34

Ինչպես արդեն տեսնում եք $m=$m+1, դա նույն գրելաձևն է C-ում, սակայն այնտեղ մենք կրճատ ենք գրել՝ m++։ Երբ while ցիկլի միջի պայմանը այլևս չի կատարվում ծրագիրը դուրս է գալիս և տպում է ավարտ։ C-ում չենք կարող տեսնել ցիկլի աշխատանքը, այն կատարվում

է շատ արագ, սակայն Կրիայում այն դեղինագույն կուրսորով տող առ տող ցույց է տալիս ցիկլի աշխատանքը(կարող ենք գործարկումը դանդաղեցնել կամ արագացնել)։

Երբեմն անհրաժեշտ է լինում ծրագիրը արտածել գոնե մեկ անգամ և հետո այն մուտք գործի ցիկլի մեջ։ Դրա համար այլագրում անհրաժեշտ է ավելացնել do հրամանը։

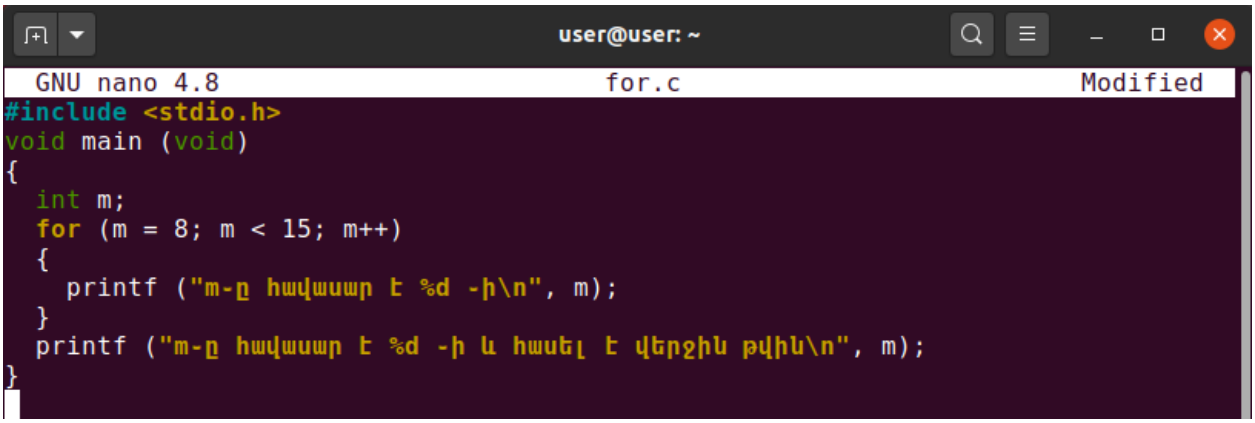
do-while – անել մինչ



Նկար 35

For ցիկլ

For ցիկլը օգտագործում ենք այն դեպքում, երբ ցանկանում ենք պայմանը կատարվի n անգամ m պայմանի առկայությամբ։ Այսինքն այն ներառում է նախորդ բոլոր ցիկլերը իր մեջ։ Մի կողմից լավ է մեկ ցիկլի հրամանով կարճ գրել խնդրի պայմանը, սակայն այն կարող է դանդաղեցնել խնդրի գործարկումը, ուստի ամեն տեղ այն կիրառելը օպտիմալ չէ։

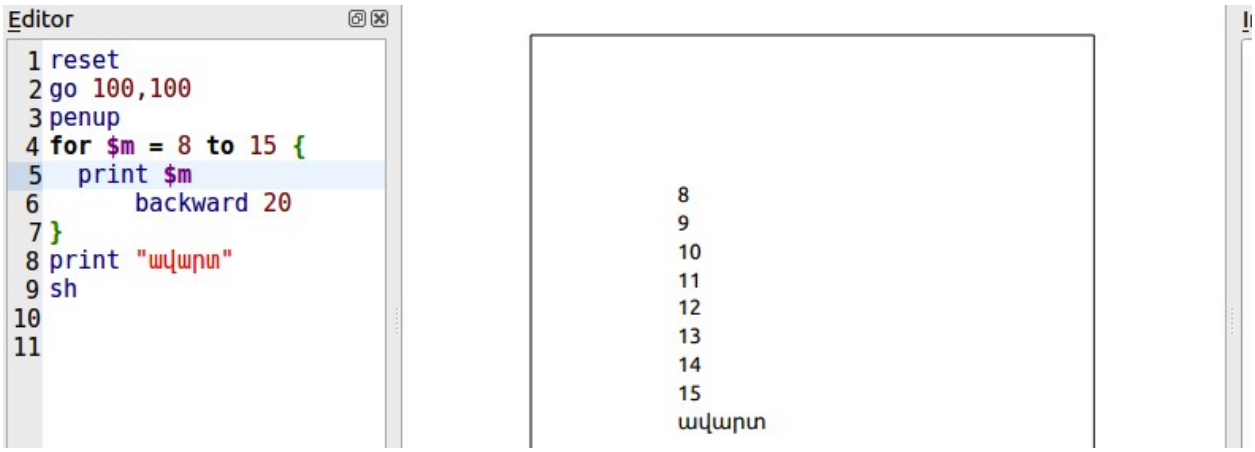


```
user@user:~$ gcc for.c -o Պատասխան.out
user@user:~$ ./Պատասխան.out
m-ը հավասար է 8 -ի
m-ը հավասար է 9 -ի
m-ը հավասար է 10 -ի
m-ը հավասար է 11 -ի
m-ը հավասար է 12 -ի
m-ը հավասար է 13 -ի
m-ը հավասար է 14 -ի
m-ը հավասար է 15 -ի և հասել է վերջին թվին
user@user:~$
```

Նկար 36

Այստեղ for ցիկլի մեջ ունենք 3 պայման, որոնք տարանջատված են կետ-ստորակետով(նախնական արժեք, փորձարկում և արժեքի ավելացում)։

Խնդիրը դիտարկենք նաև Կրիայի միջավայրում՝



Նկար 37

Ցիկլի այլագրի ոճը մի փոքր այլ է, բայց պարզ է և հեշտությամբ կարող ենք կատարել համեմատություններ։

Switch կամ փոխարկվող հրաման

Երբ պայմանների քանակը մեկից ավել է, օգտագործում են else-if հրամանը։

```
  GNU nano 4.8                    elseif.c
#include <stdio.h>
void main (void)
{
  int m = 7;
  if (m == 7)
  {
    printf ("m-ը հավասար է 7-ի\n");
  }
  else if (m == 8)
  {
    printf ("m-ը հավասար է 8-ի\n");
  }
  else
  {
  printf ("m-ը մեծ է 8-ից\n");
  }
}
```

```
user@user:~$ gcc elseif.c -o Պատասխան.out
user@user:~$ ./Պատասխան.out
m-ը հավասար է 7-ի
user@user:~$
```

Նկար 37

Եթե պայմանները շատանան, ապա այսպիսի լուծումը այլևս չի լինի խելամիտ, քանի որ շատ երկար կլինի: Այդ դեպքում պետք է օգտագործել switch հրամանը:

```
user@user: ~
  GNU nano 4.8                          switch.c
#include <stdio.h>
void main (void)
{
  int m = 7;
  switch (m)
  {
    case 7 : printf ("m-ը հավասար է 7-ի\n");
    break;
    case 8 : printf ("m-ը հավասար է 8-ի\n");
    break;
    default : printf ("m-ը մեծ է 8-ից\n");
    break;
  }
}
user@user:~$ gcc switch.c -o Պատասխան.out
user@user:~$ ./Պատասխան.out
m-ը հավասար է 7-ի
user@user:~$
```

Նկար 38

Այսինքն խնդրի արդյունքը ստացվեց նույնը ինչ if և else if հրամանների դեպքում: Հրամանը սկսվում է switch-ով և հայտարարագրում ենք m փոփոխականը: Այստեղ case հրամանով նշում ենք, որպես «դեպք» կամ այն դեպքում եթե արժեքը 7 է, ապա խնդիրը լուծված է և տպել..., իսկ break-ով անջատում ենք նախորդ տողի գործարկումը և շարունակում նոր տողինը: Default կամ լռելյայն, եթե վերոնշյալ երկու դեպքից ոչ մեկին չի համընկնում արժեքը, ապա տպել, որ այն մեծ է 8-ից:

Համեմատական օպերատորների կարճ աղյուսակ՝

Օպերատոր	Նկարագրություն	Գրելու օրինակ
==	Ստուգում է օպերանդների (հավասարման աջ և ձախ կողմերի) արժեքների հավասարությունը: Եթե **հավասար են**, ապա պայմանը դառնում է ճշմարիտ:	Ա == Բ
!=	Ստուգում է օպերանդների արժեքների անհավասարությունը: Եթե **հավասար չեն**, ապա պայմանը դառնում է ճշմարիտ:	Ա != Բ
>	Ստուգում է ձախ կողմի **օպերանդը աջից մեծ լինելու պայմանը**: Եթե մեծ է, ապա	Ա > Բ

	պայմանը դառնում է ճշմարիտ:	
<	Ստուգում է ձախ կողմի **օպերանդը աջից փոքր լինելու պայմանը**: Եթե փոքր է, ապա պայմանը դառնում է ճշմարիտ:	Ա < Բ
>=	Ստուգում է ձախ կողմի **օպերանդը աջից մեծ կամ հավասար լինելու պայմանը**: Եթե մեծ կամ հավասար է, ապա պայմանը դառնում է ճշմարիտ:	Ա >= Բ
<=	Ստուգում է ձախ կողմի **օպերանդը աջից փոքր կամ հավասար լինելու պայմանը**: Եթե փոքր կամ հավասար է, ապա պայմանը դառնում է ճշմարիտ:	Ա <= Բ

Աղյուսակ 1

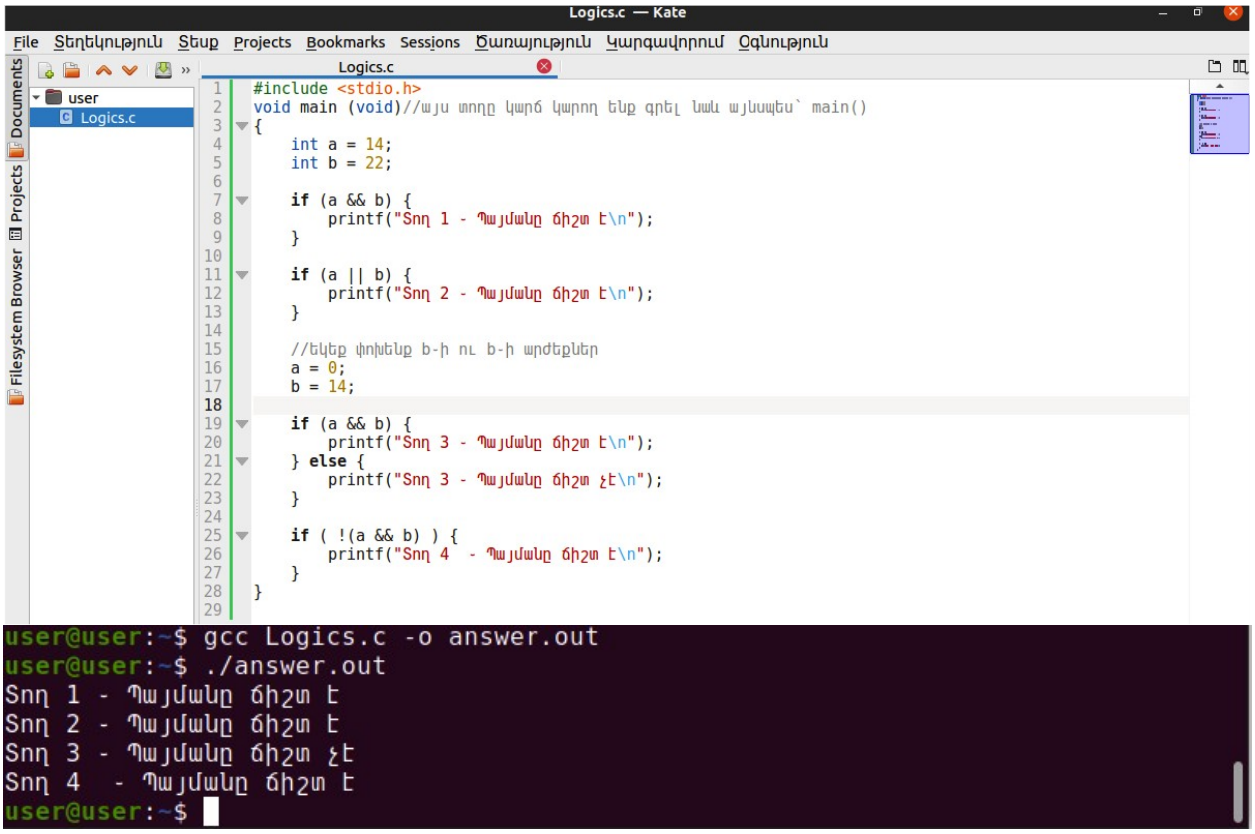
1.8. Տրամաբանական օպերատորներ

Տրամաբանական օպերատորները օգտագործվում են որոշելու տրամաբանությունը փոփոխականների կամ աժեքների միջև: Կից աղյուսակով ներկայացնենք դրանց երեք տեսակները՝

Օպերատոր	**Նկարագրություն**	**Գրելու օրինակ**
&&	Կոչվում է տրամաբանական **AND** (և) օպերատոր: Երբ երկու օպերանդները զերո չեն, ապա պայմանը դառնում է ճշմարիտ:	(Ա && Բ)
||	Կոչվում է տրամաբանական **OR** (կամ) օպերատոր: Երբ երկու օպերանդները զերո չեն, ապա պայմանը դառնում է ճշմարիտ:	(Ա || Բ)
!!	Կոչվում է տրամաբանական **NOT** (ոչ) օպերատոր: Եթե պայմանի ճիշտ է, ապա NOT օպերատորը կդարձնի այն սխալ:	! (Ա && Բ)

Աղյուսակ 2

Գրենք մի օրինակ և այս անգամ ծրագիրը հավաքենք kate միջավայրում: Kate-ը տեքստային/կոդավորման GUI խմբագրիչ է, այն հասանելի է ազատ արտոնագրով, ներառված է KDE փաթեթի մեջ և կարող եք ներբեռնել այստեղից ՝ kate-editor.org/get-it/:

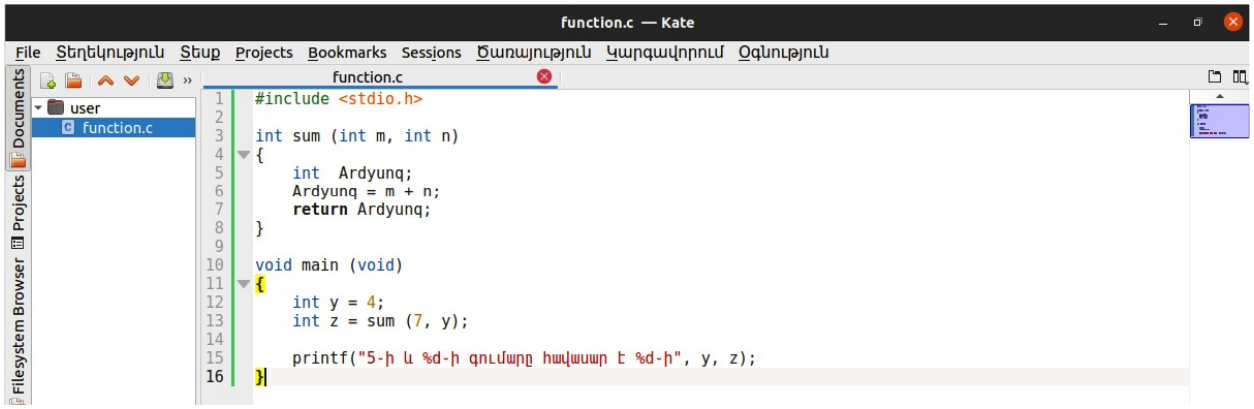


Նկար 39

1.9. Ֆունկցիաներ

Մաթեմատիկայից գիտենք՝ երբ ցանկանում ենք ցույց տալ մի փոփոխականի կախումը մյուսից, ապա օգտագործում ենք ֆունկցիա հասկացությունը: Բացի դրանից, ֆունկցիայի միջոցով կոդը բաղդատում ենք ավելի փոքր կտորների: Մինչ այս պահը մենք օգտագործում էինք main, printf ֆունկցիաները, որոնք պարզ խնդիրներ լուծելու համար են: Սակայն իրական նախագծերում օգտագործվում են տասնյակ կամ հարյուրավոր տող պարունակող կոդեր, հետևաբար ֆունկցիան, պետք է բաժանել փոքր մասերի:

Տերմինալում ստեղծենք մի նոր նիշք function.c ՝ touch function.c և խմբագրենք kate խմբագրիչով ՝ kate function.c:



Նկար 40

```
user@user:~$ touch function.c
user@user:~$ kate function.c
user@user:~$ gcc function.c -o answer.out
user@user:~$ ./answer.out
7-ի և 4-ի գումարը հավասար է 11
-huser@user:~$
```

Նկար 41

Ինչպես տեսնում եք՝ օգտագործեցինք sum և main ֆունկցիաները։ Կառուցվածքային առումով նրանք նման են իրար՝ ֆունկցիայի կողմից հետ վերադարձվող արժեքը, ֆունկցիայի անունը, ֆունկցիայի արգումենտը, որին հաջորդում է ձևավոր փակագծերի մեջ առկա սցենարը։

```
3  int sum (int m, int n)
4  {
8  }
9
10 void main (void)
11 {
16 }
```

Նկար 42

Ardyuոզ ֆունկցիան տեղային փոփոխական է, որը մենք ենք սահմանել և գտնվում է ձևավոր փակագծերի մեջ։ Տեղային նշանակում է, որ այն կարող է օգտագործվել միայն տվյալ ֆունկցիայի շրջանակներում։ Return հրամանով հետ ենք վերադարձնում խնդրի արդյունքը։

int z = sum (7, y) -ով main ֆունկցիայի կանչ ենք կատարում։

Արտաքին հղումներ

wpshopmart.com/list-of-all-famous-software-written-in-c/
en.wikipedia.org/wiki/C_(programming_language)
www.tutorialspoint.com/cprogramming/