

C ilə

Proqramlaşdırma Dünyasına Giriş

2-ci buraxılış

Şükür Hüseynov

Müəllif haqqında

Şükür Hüseynov, 1996-cı ildə Bakı şəhərində dünyaya gəlmişdir. 2014-cü ildə, Bakı şəhərində yerləşən 55 saylı tam orta məktəbi bitirmişdir və həmin il Azərbaycan Dövlət Neft və Sənaye Universitetinin Elektronika, Telekommunikasiya və Radiotexnika mühəndisliyi ixtisasına daxil olmuşdur. 2018-ci ildə universiteti başa vurmuşdur. Müəllif, 14-15 yaşlarından etibarən proqramlaşdırma ilə məşğuldur. İndiyə qədər Veb və stolüstü proqramlaşdırma, həvəskar səviyyədə isə mobil, oyun və qurğu proqramlaşdırması ilə məşğul olmuşdur. Daha çox stolüstü, oyun və sistem proqramlaşdırmasına maraq duymaqladır.

Müəllif, Proqramlaşdırmadan əlavə Psixologiya, Poeziya və Musiqi ilə də məşğul olmaqladır. Bu kitabdan əlavə elektron variantda yayımlanan həvəskar Php kitabının və bir şeir kitabının müəllifidir.

Əlavə olaraq qeyd edək ki, “Şükür Hüseynovla Proqramlaşdırma” adlı Youtube kanalımızda proqramlaşdırmaya aid videolar yayımlanmaqdadır. Videolar arasında kitabdakı bəzi mözular da yer almaqladır.

Əlaqə: Shukur.Huseynov.1996@mail.ru

Website: ShukurHuseynov.com

Mündəricat

Proqramlaşdırma – Sərhədsiz Dünya.....	5
Proqramlaşdırma dünyasına giriş.....	7
Printf funksiyası.....	10
Printf ilə ədədlər və hesablamalar.....	12
Şərhlər.....	15
Dəyişənlər.....	16
Scanf () funksiyası.....	19
Şərtlər.....	20
Digər dəyişən tipləri.....	33
Sabitlər.....	37
Çoxluqlar.....	38
Dövrələr.....	43
System funskiyası.....	49
“Windows.h” kitabxanası.....	51
For operatoru.....	57
Do While Operatoru.....	59
Funksiyalar.....	61
Fayllarla iş.....	64
Kitabxana yaratmaq.....	68
Struct tiplər.....	71
Typedef ifadəsi.....	74
Yeni dəyişən tipi.....	75
Makroslar.....	77

Yaddaş idarəetməsi və ünvan dəyişənləri.....	79
String.h kitabxanası.....	83
C++ dilinə keçid.....	90
C/C++ və HTML.....	103
Qrafiki interfeysə giriş(C++ Builder).....	105
Cihaz proqramlaşdırılması.....	130
Proqramlaşdırma məsələləri.....	132
Linuxa giriş.....	147
OpenGL və oyun proqramlaşdırması.....	151
Fon rənginin dəyişdirilməsi.....	155
OpenGL ilə fiqurların çəkilməsi.....	157
Düz xəttin çəkilməsi.....	158
Düzbucaqlının çəkilməsi.....	161
Üçbucağın çəkilməsi.....	164
Daha mürəkkəb çoxbucaqlı fiqurlar.....	167
Dairənin çəkilməsi.....	169
Çəkilən fiqurların rənglənməsi.....	176
Pəncərədə yazıların yazılması.....	178
Klaviatura əməliyyatları.....	180
Maus əməliyyatları. Mausun klik funksiyası.....	186
Maus əməliyyatları. Mausun hərəkət funksiyası.....	191
Tam Ekran.....	193
Zaman aralığı ilə əməliyyatlar.....	195
Kiçik oyunun yaradılması.....	200

Kitabın sonu.....	213
--------------------------	------------

Proqramlaşdırma – Sərhədsiz Dünya

İlk öncə “proqramlaşdırma nədir?” Sualını cavablayaq. Proqramlaşdırma sərhədsiz, sonu olmayan bir dünyadır. Bəlkə də proqramlaşdırmanı dünyanın 8-ci möcüzəsi saymaq olar. Burada sərhəd yoxdur, xəyal etdiyiniz hər şeyi virtual olaraq edə bilərsiniz. İstərsəniz xəyalınızdakı oyunu hazırlayarsınız, istərsəniz xəyalınızdakı robotu düzəldərsiniz, istərsəniz də bir şirkətin proseslərini avtomatlaşdırarsınız. İstərsəniz Hacker olarsınız, istərsəniz təhlükəsizlik mütəxəssisi olarsınız. Ən əsas məsələ xəyalınızdakı hər şeyi burada virtual olaraq edərsiniz.

50 il öncə kim inanardı ki, hesablamalar üçün yaradılmış komputerdə bu gün GTA 5, Point Blank kimi oyunlar yaradılacaq? Kim inanardı ki, bir robot insanı üzdən tanıya biləcək, tərcüməçi rolunda çıxış edəcək?

Bunları reallaşdıran ən əsas vasitə proqramlaşdırmaadır. Burada xəyal etdiyiniz hər şeyi reallaşdırma bilərsiniz. Proqramlaşdırmanı 8-ci möcüzə kimi görməyimin bir səbəbi də budur elə.

Hədəfiniz bəlkə normal bir proqramçı olmaqdır, bəlkə oyun proqramçısı olmaqdır, bəlkə hacker olmaqdır, bilmirəm, ancaq bildiyim bir şey var ki, hansını seçirsiniz seçin, ilk öncə proqramlaşdırmanın təməllərini, proqramlaşdırma məntiqini öyrənməlisiniz. Proqramlaşdırmada bəzi yerləri öyrənərkən bəlkə biraz sıxılırsınız, ürəyiniz darıxar, “bu nəyə lazımdır axı?” dediyiniz yerlər olar bəlkə, amma səbrlə, təmkinlə öyrənməyə davam etsəniz sonda nəticəni görə biləcəksiniz. Mən proqramlaşdırma öyrənməyə başlayarkən sıxıldığım, başa düşmədiyim yerlər olurdu, bezirdim bəzən. O zaman elədiyim şey, komputer başınnan qalxıb xəyal qurmaq idi. Xəyal edirdim, gələcəkdə quracağım oyunu xəyal edirdim, quracağım proqramlaşdırma şirkətini xəyal edirdim, bu məni həvəsləndirirdi, bir dəfə sıxılısam da yenidən davam edirdim. Çünki sonda xəyallarım var idi. Bilirdim ki, xəyallarıma çatmaq üçün öyrənməliyəm, sıxılısam da öyrənməliyəm, 10 dəfəyə, 20 dəfəyə başa düşsəm belə öyrənməliyəm. 1 ilə başa düşdüyüm mövzular olub. Hələ də tam anlamadığım mövzular var.

Sadəcə qarşıya məqsəd qoymaq, xəyal qurmaq, xəyallara çatmaq üçün var-güclə çalışmaq lazımdır. Proqramlaşdırmanın təməllərini öyrənərkən nə vaxt sıxıldınız, komputerin başından qalxın, qulaqcığı taxın, ən sevdiyiniz musiqini qoşun, xəyal qurmağa başlayın. Gələcəkdə nə etmək istəyirsiniz onu xəyal edin, quracağınız oyunun necə olacağını düşünün, xəyallarınızdan zövq alın və yenidən

öyrənməyə davam edin. Bir mövzunu bu kitabda anlamasanız başqa kitablara baxın, video portallara baxın mütləq bir yerdə anlayacaqsınız. Ən əsası özünü və xəyallarınıza inanın, səbrli şəkildə, var-gücünüzlə çalışın. Var-gücünüzlə çalışsanız bir gün mütləq istəklərinizə çatacaqsınız.

Proqramlaşdırma öyrənən şəxslərin verdiyi suallardan biri də proqramlaşdırma üçün riyaziyyatın lazım olub-olmamağıdır. Belə deyim ki, riyaziyyatı zəif bilsəniz belə, bunu maneə kimi görməyə ehtiyac yoxdur. Riyaziyyatı zəif bilsəniz belə proqramlaşdırma öyrənməyə başlaya bilərsiniz. Ancaq xatırladaq ki, çox böyük layihələrdə riyazi biliklərə ehtiyac duyula bilir. Ancaq, o, dediyimiz kimi böyük layihələrdir. Məsələn, oyunlarda şübhəsiz ki, riyazi biliklərdən, 2d, 3d koordinat sistemlərindən, hesablamalardan və hətta fizika qanunlarından belə istifadə olunur. Ancaq kiçik və bəzi orta layihələrdə sadə riyazi biliklər kifayət edir. Ancaq hələ ki, riyaziyyatı bir kənara qoyaq və proqramlaşdırmanın təməllərini, proqramlaşdırma məntiqini öyrənməyə çalışaq. Kifayət qədər böyük layihələr hazırladıqda(bu halda artıq peşəkar proqramçı olmuş olarsınız), riyaziyyata ehtiyac duyularsa lazım olan nəzəriyyələri öyrənərsiniz və ya bir riyaziyyatçıdan kömək alarsınız, bu qədər sadə. Onsuzda riyaziyyat bir çoxunun sevmədiyi sahədir, oxuculara sevmədiyi şeydən danışib maraqdan salmayaq.

Öyrənmə prosesi zamanı çətinliklərlə qarşılaşdıqda elə birbaşa əlaqə vasitələrimlə mənə müraciət edə bilərsiniz. Öyrənmə zamanı sizə əlimdən gələn köməkliyi etməyə çalışacam. Sizə tək virtual olaraq yox, tam təmənnasız realda da köməklik göstərə bilərəm.

İndi isə qulaqcığı taxın, ən sevdiyiniz musiqilərdən birini qoşun, gəzməyə çıxın və yenidən xəyallarınızı qurun. İstədiyiniz hər şeyi beyninizdə canlandırın, eləmək istədiklərinizi müəyyənləşdirin, özünü və inanın və xəyallarınızdan zövq alın. Ondan sonra isə gəlin artıq proqramlaşdırmanı öyrənməyə başlayaq :)

Proqramlaşdırma dünyasına giriş

Bir proqramı, mobil tətbiqi, veb saytı, robotu, oyunu, virusu proqramlaşdırmaq üçün proqramlaşdırma dillərinə ehtiyac duyulur. Bir çox proqramlaşdırma dilləri mövcuddur. Hər dilin də demək olar özünə görə üstünlüyü var. Məsələn, oyun proqramlaşdırmasında ən çox C++ dili istifadə olunur, sürətli dildir. Veb-proqramlaşdırmada Php, Asp.net çox istifadə olunur. Veb-saytların bir çoxu Php dilində yazılıb. Mobil proqramlaşdırmada Java, Kotlin dilləri çox istifadə olunur. Bizim məqsədimiz sırf proqramlaşdırma dilini yox, proqramlaşdırmanı öyrənmək olmalıdır. Çünki dil sadəcə alətdir və hər platformanın da özünə uyğun dili var. Eyni zamanda hər dövrün özünə uyğun dili olur. Məsələn, əvvəl Azərbaycanda tələb olunan əsas dillərdən biri Delphi idisə, indi elə də tələb olunmur. Bu gün ən çox istifadə olunan, bir dildirsə, sabah başqası ola bilər. Onun üçün də əsas məsələ proqramlaşdırma dilini yox, proqramlaşdırmanı öyrənməkdir. Təbii ki, proqramlaşdırmanı da proqramlaşdırma dili vasitəsilə öyrənəcəyik.

Biz proqramlaşdırmanı C dili üzərində öyrənəcəyik. İstifadə olunan bir çox dil də C dilinə oxşardı. C bilən üçün Php, Java öyrənmək də çox rahat olacaq. Facebookun qurucularından birinin Php dilini 1 həftəyə öyrəndiyini demək burada yerinə düşər. Yəqin ki, əsas proqramlaşdırma məntiqini bilirdi və yeni dil öyrənmək onun üçün rahat idi.

Deməli olduğumuz şeylərdən biri də C və C++ dillərinin fərqidir. Adları oxşardır, özləri də çox oxşardır. Ancaq C++ dili, C dilinin inkişaf etmiş formasıdır. C++ dilində obyekt yönümlü proqramlaşdırma mümkündür, C dilində isə yox. C dilini öyrəndikdən sonra C++ dilinin elementlərinə də baxacağıq.

Proqramlaşdırmaya başlaya bilməyimiz üçün bizim C kodlarını komputer dilinə çevirəcək bir proqrama ehtiyacımız var. Bu proqram da kompilyator adlanır. C kodunun komputer dilinə çevrilməsi də kompilyasiya adlanır. Komputer ikili kodlarla işləyir, yəni 0 və 1 ilə. Kompilyator da bizim C kodunu 0-1 koduna çevirir. Biz burada Windows əməliyyat sistemi üçün Dev C++ proqramından istifadə edəcəyik. Bunun üçün Dev C++ proqramını yükləməli və kompüterə quraşdırmalıyıq. Proqramın yüklənməsi və quraşdırılmasını izah edək.

1) İlk olaraq proqramı bu səhifədən yükləyin:

<https://sourceforge.net/projects/orwelldevcpp/>

Google üzərində “Dev c++ download” yazdıqda da bu səhifə çıxacaq. Yüklədikdən sonra yüklədiyimiz proqrama daxil olaq. Digər Windows proqramlarında olduğu kimi burada da açılacaq son pəncərəyə kimi Next düyməsinə klik edək. Sonda proqram kompüterə quraşdırılmış olacaq. Sonda Ok düyməsinə klik etdikdən sonra proqram açılacaq. Növbəti olaraq yuxarıdakı menyudan File->New->Projects seçək. Açılacaq yeni pəncərədə Console Application seçək və Ok düyməsinə klik edək. Bundan sonra kod yazmaq üçün sahə açılacaq. Kodlarımızı bu sahəyə yazacağıq.

İlk kodumuz ekrana “Hello world!” ifadəsini çap edən proqram olacaq. Niyə sırf bu ifadəni çap edirik deyə soruşsanız, proqramlaşdırmaya yeni başlayarkən ilk olaraq “Hello world!” ifadəsini çap etdirmək bir ənənə halını almışdır və biz də bu ənənəyə sadıq qalırıq. Proqram kodu aşağıdakı şəkildədir:

```
#include <stdio.h>

int main(){

printf("Hello world!");

getchar();

return 0;

}
```

Proqramı kompilyasiya etmək üçün F9 düyməsinə klik edək. Bundan sonra yazdığımız proqramı Desktop üzərində yadda saxlayaq. Proqramı yadda saxladığımız yerə diqqət eməliyik, çünki, növbəti dəfələrdə yadda saxladığımız proqram kodlarına yenidən daxil olmağa ehtiyac duya bilərik. Proqramı yadda saxladıqdan və kompilyasiya etdikdən sonra “Hello world!” ifadəsi ekrana çıxacaq.



Kodun izahına keçək. Bir çox əməliyyatlar funksiyalar vasitəsilə yerinə yetirilir. Yazını ekrana çap etmək funksiyası da printf funksiyasıdır. Bu funksiya üçün yazılmış yazını ekrana çap etdirir. Yazı olduğu üçün funksiyanın içində dırnaq işarələri arasında yazılır. Funksiya bitdikdən sonra sətirin sonunda ";" işarəsi qoyulur. Eyni zamanda funksiyalar kitabxanaların daxilində yerləşir. Printf funksiyası da stdio.h kitabxanasında yerləşir. İlk sətirdə include əmri ilə stdio.h kitabxanasını proqrama daxil edirik. İlk olaraq biraz qəliz gəldisə narahat olmayın, gələcəkdə daha yaxşı anlayacaqsınız. Hələ ki, sadəcə printf funksiyasına baxmağınız və bu funksiyanın bir ifadəni çap etmək üçün olduğunu bilməyiniz kifayətdir.

Yazdığımız proqramı eyni əməliyyat sistemli başqa bir komputera köçürüb orada da işə sala bilərik. Hal-hazırkı proqram qara ekranda açılsa da, gələcəkdə hazırlayacağımız digər proqramlar daha yaxşı bir görünüşə malik olacaq.

Printf funksiyası

Yuxarıda dediyimiz kimi, printf funksiyası hər-hansı bir ifadəni çap etmək üçün istifadə olunur. Proqram nəticəsində alınmış nəticəni istifadəçiyə göstərmək üçün bu funksiya vacibdir. Məsələn, təsəvvür edin ki, proqramla mürəkkəb bir hesab əməliyyatı aparırıq, amma, onu çap etmirik və istifadəçi görmür. İstifadəçiyə isə alınmış nəticə lazımdır. Ona görə bu funksiya mütləqdir.

Yuxarıda yazdığımız kodu bir daha nəzərdən keçirək.

```
#include <stdio.h>

int main(){

printf("Hello world!");

getchar();

return 0;

}
```

Gördüyümüz kimi, burada çap olunmuş ifadə yazı olduğu üçün, funksiyanın içində dırnaq işarələri arasında yazırıq. Diqqət etsəniz burada adi mətn redaktorunda olduğu kimi enterlə yeni sətirə keçə bilmirik. İfadə daxilində yeni sətirə keçmək üçün “\n” simvollarından istifadə olunur. Məsələn:

```
#include <stdio.h>

int main(){

printf("Birinci setir \n İkinci setir");

getchar();

return 0;

}
```

Nəticədə “\n” ifadəsi ilə yeni sətirə keçmiş oluruq. Burada “\n” adi yazı kimi çap olunmur və yeni sətiri bildirir. Gördüyümüz kimi, yazı daxilində boşluqlar normal qaydada, yeni sətir isə “\n” ifadəsi ilə yazılır.

Boşluq və yeni sətirə keçməklə müxtəlif fiqurlar yarada bilərik. Bunun üçün boşluq və yeni sətiri sadəcə uyğun yerlərdə qoymaq lazımdır.

```

#include <stdio.h>

int main(){

printf("*****\n");

printf("*    *\n");

printf("*    *\n");

printf("*    *\n");

printf("*    *\n");

printf("*****\n");

getchar();

return 0;

}

```

Gördüyümüz kimi, sadəcə ulduzlardan istifadə edərək kvadrat yaratmış olduq. Birinci və sonuncu sətirlər 10 ədəd ulduz istifadə edirik. Ortadakı sətirlərdə isə, sadəcə ilk və son simvol olaraq ulduz, digər simvolların yerinə isə boşluq istifadə edirik. Nəticədə istədiyimiz fiqur yaranmış olur.

İndi isə gəlin bu ulduzlarla böyük bir “A” hərfi yazaq:

```

#include <stdio.h>

int main(){

printf("    *    \n");

printf("    * *    \n");

printf("    * *    \n");

printf("    * *    \n");

printf("    * *    \n");

printf("    * *    \n");

printf(" ***** \n");

printf(" *          * \n");

```

```

printf(" *      * \n");
printf(" *      * \n");
printf("*      *\n");
getchar();
return 0;
}

```

Gördüyünüz kimi, böyük bir “A” hərfi yazdıq. İndi isə “T”, “U”, “İ”, “Z”, “N” hərflərini özünüz yazmağa cəhd edin.

Bu xüsusiyyətlərdən istifadə edərək gələcəkdə kiçik oyunlar, animasiyalar belə yazıla bilər.

Printf ilə ədədlər və hesablamalar

Funksiya ilə ədədləri də çap edə bilərik. Ancaq, ədədlərin çap olunması qaydası biraz fərqlidir. Koda baxaq:

```

#include <stdio.h>

int main(){
printf("Eded: %d",5);
getchar();
return 0;
}

```

Printf funksiyasına iki parametrlə ötürülür. Birinci parametrlə "Eded: %d", ikinci parametrlə isə 5 rəqəmi. Nəticədə birinci ifadədəki “%d” hissəsi 5 ilə əvəz olunur.

Əgər bir neçə ədəd çap etmək istəsək, onda, birinci ifadənin içində işlətmək istədiyimiz ədəd qədər “%d” yazmalıyıq.

```

#include <stdio.h>

```

```
int main(){
printf("Eded: %d %d",5, 6);
getchar();
return 0;
}
```

Nəticədə birinci “%d” hissəsi 5 ilə, ikinci “%d” hissəsi isə 6 ilə əvəz olunacaq. İstəyimizə görə ədədlərin sayın artırma da bilərik.

Ədədləri sadəcə çap etdiyimiz kimi, onlar üzərində hesab əməlləri apararaq da onları çap edə bilərik. Toplama əməli üçün “+” işarəsindən, çıxma əməli “-” üçün işarəsindən, vurma əməli üçün “*” işarəsindən, bölmə əməli üçün isə “/” işarəsindən istifadə olunur. Sadə bir nümunə göstərək:

```
#include <stdio.h>

int main(){
printf("Eded: %d",10+5);
printf("\n");
printf("Eded: %d",10-5);
printf("\n");
printf("Eded: %d",10*5);
printf("\n");
printf("Eded: %d",10/5);
getchar();
return 0;
}
```

Nəticədə alınmış ədəd çap olundu. Hesablamada iki ədəddən istifadə olunsada, nəticədə bir ədəd alındığı üçün, bir ədəd “%d” işarəsindən istifadə edirik. Eyni zamanda, daha mürəkkəb hesablamalar apara və bir printf funksiyası ilə bir neçə

hesablamanın nəticəsini göstərə bilərik. Bunun üçün bir neçə “%d” işarəsindən istifadə edəcəyik.

```
#include <stdio.h>

int main(){

printf("Eded: %d",10*5+6-2);

printf("\n");

printf("Eded: %d, %d, %d", 10+5, 20*5, 20/2);

getchar();

return 0;

}
```

Riyaziyyatda olduğu kimi, burada da ilk öncə vurma və bölmə, sonra isə toplama və çıxma əməlləri yerinə yetirilir. Məsələn:

```
#include <stdio.h>

int main(){

printf("Eded: %d",10+6*2);

getchar();

return 0;

}
```

Burada $6*2$ hesablanaraq 10-un üzərinə gəlinir. Əgər biz əvvəlcə $10+6$ ifadəsini hesablayıb sonra 2-ə vurmaq istəsək, $10+6$ ifadəsini mötərizə daxilində yazmalıyıq.

```
#include <stdio.h>

int main(){

printf("Eded: %d",(10+6)*2);

getchar();

return 0;
```

```
}
```

Nəticədə cavab 32 alınır.

Bəzən tam bölünmədən alınmış qalıqı tapmağa ehtiyac olur. Məsələn, ədədin cüt ədəd olub-olmadığını yoxlamaq üçün qalıqlı bölməyə ehtiyac duyulur. Fərz edək ki, ədədimiz 32-dir. Bu ədədin cüt olub-olmamağın yoxlamaq üçün 32-i 2-ə bölürük və qalıq 0 alınır. Qalıq 0-dırsa deməli ədəd cüt ədəddir. Əgər qalıq 1 olarsa, deməli ədəd tək ədəddir. Qalıqı tapmaq üçün “%” işarəsindən, bölmədən alınmış tam hissə üçün isə, elə bölmə üçün istifadə etdiyimiz “/” işarəsindən istifadə edəcəyik.

```
#include <stdio.h>

int main(){

printf("Tam hisse: %d",20/6);

printf("\n");

printf("Qaliq hisse: %d",20%6);

printf("\n");

getchar();

return 0;

}
```

Nəticədə tam hissə və qalıq hissə çap olunur. Növbəti mövzularda onluq bölməni və digər mürəkkəb riyazi funksiyaları da görəcəyik.

Şərhlər

Bəzən proqramlar uzun olduqda onları sonradan oxumaq çətinləşir. Bunun üçün proqram daxilində şərhərdən istifadə olunur. Şərhlər proqramda dəyişikliyə yol açmır, sadəcə, bizim sonradan kodu anlamağımızı asanlaşdırır. Bir sətirdə şərh yazmaq üçün, // işarələrindən istifadə olunur.

```
#include <stdio.h>

int main(){
```



```
printf("Eded: %d",5); // Burada 5 çap olunacaq.  
  
getchar();  
  
return 0;  
  
}
```

Çoxsətirli şərtləri yazmaq üçün isə, /* və */ işarələrindən istifadə olunur.
Məsələn:

```
#include <stdio.h>  
int main(){  
printf("Eded: %d",5);  
/*  
Burada 5 çap olunacaq.  
Şərhlər  
*/  
getchar();  
return 0;  
}
```

Dəyişənlər

İlk öncə dəyişənlərin nə olduğunu, harada istifadə olunduğunu anlamağa çalışaq. Təbiətdə, kainatda saysız-hesabsız dəyişənlərə rast gələ bilərik. Məsələn, otaqdakı temperatur bir dəyişəndir. Əvvəlcə 20 dərəcə olur, bir saat keçir 22 dərəcə olur, sonra aşağı düşür 21 dərəcə olur. Zaman keçdikcə qiyməti dəyişir, buna görə də dəyişən adlanır. Elə ola bilər ki, temperatur heç dəyişməsin, amma biz yenə də onu proqramlaşdırmada dəyişən kimi ifadə edə bilərik.

Başqa bir nümunə də göstərək. Məsələn, hər-hansı bir oyunda xarakterimizin 100 canı olur, sonra bir qəzaya düşür və canı biraz azalır və 70 olur. Sonra zaman keçir, artaraq 80 olur. Gördüyümüz kimi, burada xarakterin canı dəyişəndir və tez-tez dəyişir. İstifadəçinin oyunda topladığı xalları da nümunə göstərə bilərik. İstifadəçinin əvvəl 0 xalı olur, sonra biraz oynayır və xalı artır, 1 olur, sonra 5 olur, sonra 10 olur və artır. Ona görə, proqramlarda, oyunlarda dəyişənlərə ehtiyac duyulur. Dəyişənlər elə riyaziyyatda da işlənir. Məsələn, x dəyişəni, y dəyişəni və s. şəkildə yazırıq.

Dəyişənin nə olduğunu anladığımıza görə artıq proqramlaşdırmada istifadə qaydasına baxa bilərik. Bəzi proqramlaşdırma dillərində bütün dəyişənləri eyni şəkildə asanlıqla ifadə etmək olur. Amma C dilində dəyişəni elan edərkən onun tipini də təyin etmək lazım gəlir. Dəyişən müxtəlif tiplərdə ola bilər. Məsələn tam ədəd tipində. Dəyişən yalnız tam ədədlər ala bilər. Məsələn 5, 6, -3, 0, 4 və s. qiymətlərini ala bilər. C dilində tam tipli dəyişən **“int”** açar sözü ilə elan olunur. Dəyişənin adı hərflərdən rəqəmlərdən və bəzi simvollarından ibarət ola bilər. Məsələn, “temperatur” adlı bir dəyişən elan edək və onu çap edək.

```
#include <stdio.h>

int main(){

int temperatur;

temperatur=5;

printf("Temperaturun qiymeti: %d", temperatur);

getchar();

return 0;

}
```

Gördüyümüz kimi, üçüncü sətirdə int açar sözündən sonra temperatur yazdıq. Burada “int” sözü tam tipi bildirən açar söz, “temperatur” isə dəyişənin adıdır. Sətrin sonunda isə “;” qoyuruq. Dördüncü sətirdə dəyişənə “=” simvolundan istifadə edərək dəyişənə 5 qiymətini veririk və dəyişənimiz 5-ə bərabər olur. Növbəti sətirdə isə, printf funksiyası ilə dəyişəni çap edirik. Dəyişən ədəd şəklində olduğu üçün, bu dəyişəni ədədləri çap etdiyimiz şəkildə çap edirik. Birinci ifadədə “%d” yazırıq, ikincidə isə dəyişəni. Nəticədə “%d” hissəsi dəyişənin qiyməti ilə əvəz olunur.

İndi isə dəyişənlər vasitəsilə iki ədədin cəmini toplayaq. Bunun üçün 3 ədəd dəyişənimiz olacaq: a, b və c dəyişənləri. Burada a birinci ədəd, b ikinci ədəd, c isə bu ədədlərin cəmi olacaq və biz bu ədədləri toplayacağıq. Koda baxaq:

```
#include <stdio.h>

int main(){

int a,b,c;
```

```
a=7;

b=8;

c=a+b;

printf("%d",c);

getchar();

return 0;

}
```

Nəticədə 15 çap olunur. İndi isə bütün hesablama əməllərindən istifadə edək:

```
#include <stdio.h>

int main(){

int a,b,c,d,e,f;

a=15;

b=3;

c=a+b;

d=a-b;

e=a*b;

f=a/b;

printf("Ədədlərin cəmi: %d",c);

printf("\nƏdədlərin fərqi: %d",d);

printf("\nƏdədlərin hasili: %d",e);

printf("\nƏdədlərin nisbəti: %d",f);

getchar();

return 0;

}
```

Gördüyümüz kimi bir neçə dəyişən elan etdik və hər birinə müxtəlif nəticəni mənimsətdik.

Tam tiptən əlavə də bir sıra dəyişən tipləri mövcuddur. Ancaq, hələki beynimizi qarışdırmamaq üçün digər tipləri sonraya saxlayaq.

Scanf () funksiyası

Printf() funksiyası ilə yanaşı istifadə edəcəyimiz digər bir funksiya da scanf() funksiyasıdır. Əvvəlki hissədə dəyişənin qiymətini proqramı yazarkən təyin etmişdik. Ancaq dəyişənin qiyməti proqram işə düşdükdən sonra da istifadəçi tərəfindən təyin oluna bilər. Bunun üçün scanf() funksiyasından istifadə etməliyik. Funksiyanın istifadəsinə baxaq:

```
#include <stdio.h>

int main(){

int temperatur;

printf("Temperaturun qiymətini daxil edin:\n");

scanf("%d",&temperatur);

printf("Temperaturun qiymeti: %d", temperatur);

getchar();

return 0;

}
```

Proqram işə düşdükdən sonra temperatura bir qiymət daxil edirik(məsələn 5) və proqram onu printf() funksiyası ilə çap edir. Gördüyümüz kimi, scanf() funksiyasında printf() funksiyasında olduğu kimi birinci parametr olaraq «%d» hissəsini yazırıq, ikinci parametr olaraq isə dəyişənin adını. Dəyişənin adından əvvəl «&» işarəsini də yazmağımız mütləqdir.

Scanf() funksiyasını harada işlədəcəyik deyə soruşsanız sadə bir cavab verə bilərəm. Fərz edin ki, bir hesablama proqramı düzəldirsiniz. Burada hesablama aparılacaq ədədləri istifadəçi daxil etməlidir. Yəni qiymətlər proqram işə

düşdükdən sonra daxil olunacaq. Bunun üçün də scanf() funksiyasından istifadə olunmalıdır.

İndi isə scanf() funksiyasından istifadə edərək istifadəçidən iki ədəd qəbul edək və onların cəmini çap edək:

```
#include <stdio.h>

int main(){

int a,b,c;

printf("Birinci ədədi daxil edin: ");

scanf("%d",&a);

printf("\nikinci ədədi daxil edin: ");

scanf("%d",&b);

c=a+b;

printf("\nDaxil etdiyiniz ədədlərin cəmi: %d",c);

getchar();

return 0;

}
```

Nəticədə a və b dəyişənlərinin qiyməti istifadəçi tərəfindən daxil olunacaq və ədədlərin cəmi c dəyişəninə mənimsədilərək ekranda çap olunacaq.

Şərtlər

Həyatda bir çox şeyi şərtlərə əsasən edirik. Məsələn, hava çox istidirsə, yəni bu şərt ödənirsə, nazik paltarlar geyinirik. Yox əgər, hava soyuqdursa, qalın paltar geyinirik. Bunu bu şəkildə yoxlaya bilərik. Əgər hava istidirsə nazik paltar geyin, yox əgər hava soyuqdursa qalın paltar geyin. Gördüyümüz kimi, havanın soyuq və ya isti olması halında fərqli şeylər edirik. Proqramlaşdırmada da bu şəkildədir. Bir şərtin ödənməsindən asılı olaraq bir işi yerinə yetirə bilərik. Məsələn, belə bir proqram düşünek. İstifadəçidən 3+2 ifadəsinin nəticəsini soruşaq. İstifadəçi ədədi daxil etsin, sonra isə daxil etdiyi ədədi şərtlə yoxlayaq. Əgər daxil etdiyi ədəd 5

olarsa, doğru daxil etdiyini yazaq. Şerti yoxlamaq üçün **if** operatorundan istifadə edəcəyik. Bu operatorun istifadə qaydasını aşağıdakı şəkildə göstərək:

if(şərt) əməliyyat

If-dən sonra mötərizə açıq və mötərizənin daxilində şərt yazırıq. Əgər şərt ödənersə əməliyyat yerinə yetirilir. Şərt ödənməzsə həmin əməliyyat yerinə yetirilmir. Yuxarıda təsəvvür etdiyimiz proqramı yazaraq if operatorunun istifadə qaydasını göstərək:

```
#include <stdio.h>

int main(){

int a;

printf("3+2=?\n");

scanf("%d",&a);

if(a==5) printf("Təbriklər, doğru daxil etdiniz.");

getchar();

return 0;

}
```

İlk olaraq diqqət edək ki, if operatorunun içində bərabərliyi yoxlamaq üçün iki ədəd «=» işarəsini yazmızıq. Bərabərliyi yoxlayan zaman iki ədəd «=» işarəsini yazmaq lazımdır.

İlk olaraq a dəyişənini elan etdik. Sonra printf() funksiyası ilə yazını çap etdik. Sonra isə scanf() funksiyası ilə istifadəçidən a dəyişəninin qiymətini qəbul etdik. Sonra isə, if operatoru vasitəsilə a-nın qiymətinin 5-ə bərabər olub olmadığını yoxladıq. Yoxladığımız ifadənin doğru olması halında, "Təbriklər, doğru daxil etdiniz." ifadəsini çap etdik. Nəzərə çatdıraq ki, bu şəkildə yalnız if-dən sonrakı bir əməliyyat yerinə yetiriləcək. Əgər if-dən sonra iki dəfə printf funksiyasından istifadə etmiş olsaydıq, ikinci istifadə etdiyimiz printf() funksiyası şərtə tabe olmayacaqdı. Yuxarıdakı proqrama biraz dəyişiklik edək:

```
#include <stdio.h>

int main(){
```

```

int a;

printf("3+2=?\n");

scanf("%d",&a);

if(a==5) printf("Təbriklər, doğru daxil etdiniz.");

printf("\nBu ifadə şərtədən asılı olmadan çap olunacaq.");

getchar();

return 0;

}

```

Nəticədə daxil etdiyimiz ədəddən asılı olmayaraq "\nBu ifadə şərtədən asılı olmadan çap olunacaq." ifadəsi çap olunacaq

Öyrəşməyiniz üçün bir neçə ədəd bu tip proqram yazmağınız mütləqdir. Yazdığınızı silin və yenidən yazın, yaza bilməsəniz kitaba baxın, bir neçə dəfə fərqli-fərqli suallar soruşun, 3+2-nin yerinə 5-2 ifadəsinin qiymətini soruşun, otaqdakı temperaturun qiymətini soruşun və onun 25 olub olmadığını yoxlayın və ya başqa nümunələr yazın. Yaxşı anlamaq üçün mümkün qədər çox proqram yazın.

İndi isə bir proqramda bir neçə şərt yoxlayaq. İstifadəçidən bir neçə sual soruşaq və cavablar əsasında istifadəçinin nəticəsini göstərək.

```

#include <stdio.h>

int main(){

int a,b,c;

printf("3+2=?\n");

scanf("%d",&a);

printf("5-2=?\n");

scanf("%d",&b);

printf("7*2=?\n");

scanf("%d",&c);

```

```

if(a==5) printf("Təbriklər, 1-ci suala doğru cavab verdiniz.");

if(b==3) printf("Təbriklər, 2-ci suala doğru cavab verdiniz.");

if(c==14) printf("Təbriklər, 3-cü suala doğru cavab verdiniz.");

getchar();

return 0;

}

```

Burada a,b və c dəyişənlərini elan etdik. İstifadəçidən 3 sual soruşduq və suallara cavab verməsini istədik. Birinci sualın cavabı 5 idi, ona görə də a dəyişəninin 5-ə bərabər olub-olmamasını yoxladıq və ona uyğun istifadəçiyə məlumat verdik. Növbəti olaraq isə b dəyişəninin 3-ə bərabər olub-olmamağını yoxladıq və ona uyğun istifadəçiyə məlumat verdik. Sonda isə 3-cü sualın cavabı 14 olduğu üçün c-nin 14-ə bərabər olub-olmamağını yoxladıq və istifadəçiyə məlumat verdik.

Yuxarıdakı proqramda istifadəçiyə nəticə olaraq hansı suallara doğru cavab verdiyini göstərdik. İndi isə istifadəçiyə neçə suala cavab verdiyini göstərək. Bunun üçün proqramın əvvəlində bir ədəd say dəyişəni elan edirik və onun qiymətini 0 təyin edirik. Əgər birinci suala doğru cavab versə, onun qiymətini 1 vahid artırırıq. Beləliklə hələki 1 suala doğru cavab verdiyini say dəyişəni ilə bilirik. İkinci suala doğru cavab versə say dəyişənini yenə 1 vahid artırırıq. Əgər ikinci də doğru cavab versə say dəyişəninin qiyməti 2 olur. Beləliklə istifadəçi 2 suala doğru cavab vermiş olur. Növbəti olaraq üçüncü suala da doğru cavab versə sayın qiymətini 1 vahid də artırırıq və sayın sonuncu qiyməti doğru cavab sayını göstərmiş olacaq. Bunu bir proqramla göstərək:

```

#include <stdio.h>

int main(){

int a,b,c,say;

say=0;

printf("3+2=?\n");

scanf("%d",&a);

printf("5-2=?\n");

```



```

scanf("%d",&b);

printf("7*2=?\n");

scanf("%d",&c);

if(a==5) say=say+1;

if(b==3) say=say+1;

if(c==14) say=say+1;

printf("\nDoğru cavab sayı: %d",say);

getchar();

return 0;

}

```

Proqramı işə saldıqdan sonra qiymətlər olaraq 4, 3 və 14 daxil edək və proqramı analiz edək. İlk olaraq 4 daxil etdiyimiz üçün a dəyişəninin qiyməti 4, b dəyişəninin qiyməti 3 və c dəyişəninin qiyməti 14 olmuş olur. Növbəti olaraq şərtə a-nın $5 = 5$ -ə bərabər olub-olmamasını yoxlayırıq, çünki doğru cavabımız 5-dir. Ancaq a-nın qiymətini biz 4 olaraq daxil etmişik və beləliklə şərt ödənmir və nəticədə say dəyişəninin qiyməti artmır. Növbəti olaraq b dəyişəninin qiymətini 3 olaraq daxil etmişik. Bunun üçün də b-nin 3-ə bərabər olub olmamasını yoxlayırıq. Burada artıq şərt ödəndiyi üçün say 1 vahid artmış olur. Növbəti dəyişənimiz c-nin qiymətini isə 14 olaraq daxil etmişik və şərtə bu qiymətin 14 olub olmadığını yoxlayırıq. Həqiqətən də, c-nin qiyməti 14 olduğu üçün say dəyişəni 1 vahid də artır. Və nəticədə, iki sualın cavabını doğru yazdığımız üçün, say dəyişəni iki dəfə 1 vahid artdı və 2 oldu. Beləliklə say dəyişəni doğru cavabların sayını bildirmiş oldu. Sonda isə say dəyişəninini çap edərək bunu istifadəçiyə bildirdik.

İndi isə, sıra ilə 5 ədədlik, 10 ədədlik kiçik testlər tərtib edin. Testlərdə doğru cavab sayını da, yanlış cavab sayını da və eyni zamanda hansı cavabların doğru olduğunu da göstərin. Yanlış cavabların sayını tapmaq üçün ümumi test sayından doğru cavab sayını çıxmağınız kifayətdir. Məsələn, ümumi test sayı 5-dirsə, proqramın sonunda doğru cavab sayını hesabladıqdan sonra 5-dən doğru cavab sayını çıxaraq yanlış cavab sayını tapa bilərsiniz. Allınmış proqramları dostlarınızla da bölüşün və sizin test proqramı tərtib edə biləcəyinizi görsünlər :)

Yuxarıdakı nümunələrdə biz yalnız 1 şərt üçün bir əməliyyat daxil edirdik. Bəs bir şərt üçün bir neçə əməliyyat yazmaq istəyiriksə? Məsələn, həm sayı artırmaq, həm də şərt daxilində bir neçə dəfə printf funksiyası işlətmək. Bunun üçün şərtdən sonrakı əməliyyatları «{» və «}» işarələri arasında yazmalıyıq. Məsələn:

```
#include <stdio.h>

int main(){

int a,say;

say=0;

printf("3+2=?\n");

scanf("%d",&a);

if(a==5){

printf("\n Doğrudur.");

printf("\n Doğru daxil etdiniz.");

say=say+1;

}

printf("\n %d",say);

getchar();

return 0;

}
```

Gördüyümüz kimi, şərtdən sonra bir neçə əməliyyat yazmaq üçün əməliyyatları «{» və «}» işarələri arasında yazmalıyıq. Beləliklə, «{» və «}» arasında yazılan əməliyyatlar şərt doğru olduğu halda yerinə yetiriləcək.

Şərt daxilində bərabərlikdən əlavə böyük bərabər, kiçik bərabər, fərqli və bir sıra şərtləri də yoxlamaq mümkündür. Bunun üçün aşağıdakı işarələrdən istifadə edə bilərik:

> Böyükdür

< Kiçikdir

>= Böyük bərabərdir

<= Kiçik bərabərdir

!= Fərqlidir

İndi isə bu ifadələrin hamısını eyni bir proqramda istifadə edək:

```
#include <stdio.h>

int main(){
    int a,b;
    printf("Birinci ədəd:\n");
    scanf("%d",&a);
    printf("\nikinci ədəd:\n");
    scanf("%d",&b);
    if(a>b){
        printf("\na>b");
    }
    if(a<b){
        printf("\na<b");
    }
    if(a==b){
        printf("\na=b");
    }
    if(a>=b){
        printf("\na>=b");
    }
}
```

```

if(a<=b){

printf("\na<=b");

}

getchar();

return 0;

}

```

Göründüyü kimi, bir proqramda bir neçə şərtlə müxtəlif ifadələri göstərdik. Burada daxil edilən ədədlərə görə iki şərt birdən ödəyə bilər. Məsələn, 5 və 4 daxil edilsə, həm böyükdür şərti, həm də böyük bərabərdir şərti ödəyəcək və buna uyğun iki dəfə çap əməliyyatı yerinə yetiriləcək. Çünki, 5 ədədi 4-dən həm böyük, həm də böyük bərabərdir.

Bir şərtlə bərabər həmin şərtin əks halları da mövcuddur. Məsələn, daxil edilən qiymət 5 olarsa cavab düzgündür, əks hallarda isə səhvdir. Əks hallar dedikdə 5-dən fərqli qiymətləri nəzərdə tuturuq. Əks hallarda müəyyən əməliyyatları yerinə yetirmək üçün if ilə yanaşı köməkçi **else** hissəciyindən istifadə etmək lazımdır. Bu ayrıca operator deyil, if ilə bir yerdə istifadə olunur. Bu hissəcik if-in əməliyyatları bitdikdən sonra yazılır. Nümunəyə baxaq:

```

#include <stdio.h>

int main(){

int a;

printf("3+2=?\n");

scanf("%d",&a);

if(a==5){

printf("Doğru cavab yazdınız.");

}

else{

printf("Yalnış cavab verdiniz.");

}

}

```

```
getchar();  
  
return 0;  
  
}
```

Program işə düşdükdən sonra daxil edilən ədəd 5 olarsa ekrana "Doğru cavab yazdınız." çıxacaq. 5-dən fərqli daxil ediləcək bütün ədədlər üçün isə, "Yalnış cavab verdiniz." ifadəsi çap olunacaq. Gördüyümüz kimi, else hissəciyinin də istifadəsi if kimidir, else yazdıqdan sonra «{» və «}» işarələri arasında əməliyyatları yazırıq. If hissəsindəki şərt ödənmədiyi bütün hallarda else hissəsindəki əməliyyatlar yerinə yetirilir. If hissəsindəki şərt ödəndikdə isə, else hissəsinə baxılmaz.

Başqa bir kiçik nümunə də yazaq. Bu dəfə istifadəçidən iki ədəd daxil etməsini tələb edək və bu ədədlərin bərabər olub-olmamasını yoxlayaq.

```
#include <stdio.h>  
  
int main(){  
  
    int a,b;  
  
    printf("Birinci ədəd:\n");  
  
    scanf("%d",&a);  
  
    printf("\nikinci ədəd:\n");  
  
    scanf("%d",&b);  
  
    if(a==b){  
  
        printf("a və b bərabərdir.");  
  
    }  
  
    else{  
  
        printf("a və b bərabər deyil.");  
  
    }  
  
    getchar();  
  
    return 0;
```

```
}
```

Burada daxil edilən hər iki ədəd eyni olarsa, məsələn, hər ikisi 5 olarsa, "a və b bərabərdir." yazısı çap olunacaq. Bərabər olmadıqda isə, "a və b bərabər deyil." yazısı çap olunacaq.

Yuxarıdakı nümunələrdə şərt və onun əksi əməliyyatlar yazdıq. Bəs əgər iki müxtəlif halı yoxlamaq istəyiriksə? Məsələn, istifadəçidən 25 və 28 arasında bir ədəd daxil etməsini tələb edirik. İstifadəçi burada həm 26-nı daxil edə bilər, həm də 27-i. Bəs necə edək? Burada artıq **else if** hissəciyindən istifadə edəcəyik. Onun vasitəsilə ikinci bir şərti də yazacağıq. İstifadəsi if- ilə eynidir, sadəcə əvvəlinə else yazırıq. Nümunə göstərək:

```
#include <stdio.h>

int main(){

int a,b;

printf("Ədəd:\n");

scanf("%d",&a);

if(a==26){

printf("Doğru: 26");

}

else if(a==27){

printf("Doğru: 27");

}

else{

printf("Yalnış.");

}

getchar();

return 0;

}
```

Burada ilk öncə if hissəsinə baxılır. Əgər a dəyişəni 26-a bərabər olarsa, if-in daxilindəki əməliyyatlar yerinə yetirilir. Doğru olmazsa else if hissəsinə keçilir. Bu hissədə də a dəyişəninin 27-ə bərabər olub-olmaması yoxlanılır. Doğru olarsa bu hissədəki əməliyyatlar yerinə yetirilir. Doğru olmadığı halda daxil olunan qiymətdən asılı olmayaraq else hissəsinə keçir və bu hissədəki əməliyyatlar yerinə yetirilir.

Else if hissəsindən bir neçə dəfə də istifadə edə bilərik. Məsələn, ədədin 26, 27 və ya 28-ə bərabər olub-olmamasını yoxlaya bilərik. Koda baxaq:

```
#include <stdio.h>

int main(){

int a,b;

printf("Ədəd:\n");

scanf("%d",&a);

if(a==26){

printf("Doğru: 26");

}

else if(a==27){

printf("Doğru: 27");

}

else if(a==28){

printf("Doğru: 28");

}

else{

printf("Yalnış.");

}

getchar();

return 0;
```

```
}
```

Gördüyümüz kimi, else if hissəsindən iki dəfə istifadə etdik. Bundan əlavə istədiyimiz qədər istifadə edə bilərik. Bu proqramın da işləmə qaydası yuxarıda göstərdiyimiz şəkildədir.

Bundan əlavə, bir şərt daxilində bir neçə halı yoxlamağımız mümkündür. Məsələn, fərz edək ki, iki ədəd a və b dəyişənimiz var. Eyni zamanda birinci dəyişən 5-ə, ikinci dəyişən 6-a bərabər olarsa, müəyyən əməliyyatları yerinə yetirmək istəyirik. Bunun üçün bir şərt daxilində iki halı yazmalıyıq. Koda baxaq:

```
#include <stdio.h>
```

```
int main(){
```

```
int a,b;
```

```
a=5;
```

```
b=6;
```

```
if(a==5 && b==6){
```

```
printf("Ok");
```

```
}
```

```
getchar();
```

```
return 0;
```

```
}
```

Gördüyümüz kimi, həm a dəyişəni 5-ə, həm də b dəyişəni 6-a bərabər olduğu üçün şərt ödənilir və yazı çap olunur. Amma, əgər a dəyişəni 5-dən fərqli olsaydı və ya b dəyişəni 6-dan fərqli olsaydı, o zaman şərt ödənməyəcəkdi və əməliyyat yerinə yetirilməyəcəkdi. Bir şərt daxilindəki halların sayın artırma da bilərik:

```
#include <stdio.h>
```

```
int main(){
```

```
int a,b,c;
```

```
a=5;
```



```

b=6;

c=7;

if(a==5 && b==6 && c==7){

printf("Ok");

}

getchar();

return 0;

}

```

Burada da, gördüyümüz kimi hər 3 hal ödəndiyi üçün şərt doğru olur və əməliyyat yerinə yetirilir.

Şərt daxilində bütün halların doğru olması zamanı əməliyyatları yerinə yetirmək üçün «&&» simvollarından istifadə etdik. Ancaq, göstərilən hallardan ən azı birinin doğruluğu zamanı müəyyən əməliyyatlar yerinə yetirmək istəyiriksə, «|» simvollarından istifadə etməliyik.

```

#include <stdio.h>

int main(){

int a,b,c;

a=5;

b=8;

if(a==5 || b==6){

printf("Ok");

}

getchar();

return 0;

}

```

Nəticədən, göstərilən hallardan ən azı biri doğru olduğu üçün(a=5 olduğu üçün) şərt ödənilir və əməliyyat yerinə yetirilir.

Digər dəyişən tipləri

İlk başda çətinlik olmasın deyə sadəcə int tipinə baxdıq. İndi isə digər dəyişən tipləri ilə tanış olaq. Tam ədədlərdən əlavə onluq ədədlər də mövcuddur. Məsələn, 5.6, -0.7 və s. ədədlər də mövcuddur. Bu ədədlər tam ədəd olmadıqları üçün onları int tipi ilə elan edə bilmərik. Bu ədədləri elan etmək üçün **float** tipindən istifadə edə bilərik.

```
#include <stdio.h>

int main(){

float a;

a=5.5;

printf("Dəyişən: %f",a);

getchar();

return 0;

}
```

Gördüyümüz kimi, float tipli dəyişəni printf funksiyası ilə çap edərkən, «%d» hissəsindən yox, «%f» hissəsindən istifadə etməliyik.

Digər bir tip isə double tipidir. Double tipi də float tipi kimi onluq dəyişənləri elan etmək üçün istifadə olunur. Nümunə:

```
#include <stdio.h>

int main(){

double a;

a=5.5;

printf("Dəyişən: %f",a);
```

```
getchar();  
  
return 0;  
  
}
```

Printf() funksiyası daxilində double tipli dəyişəni çap edərkən, float tipində olduğu kimi «%f» hissəsindən istifadə olunur.

Daha dəqiq hesablamalar zamanı double tipindən istifadə etmək daha məqsədəuyğundur. **Double və floatın tipinin əsas fərqi, double tipinin daha dəqiq olmasıdır.** Növbəti mövzularda bu hissə daha ətraflı izah olunacaq.

Növbəti öyrənəcəyimiz dəyişən tipi **char** tipidir. Bu dəyişən tipi simvolları elan etmək üçün istifadə olunur. Nümunəyə baxaq:

```
#include <stdio.h>  
  
int main(){  
  
char a;  
  
a='z';  
  
printf("Dəyişən: %c",a);  
  
getchar();  
  
return 0;  
  
}
```

Burada dəyişənin qiyməti yalnız bir simvoldan ibarət ola bilər. Gördüyümüz kimi, printf daxilində char tipindəki dəyişəni çap etmək üçün, «%c» hissəsindən istifadə edirik.

İndi isə, gəlin istifadəçidən bir simvol daxil etməsini tələb edək və bu simvolu şərtlə yoxlayaraq müxtəlif əməliyyatlar aparaq.

```
#include <stdio.h>  
  
int main(){  
  
char a;  
  
printf("Bir hərf daxil edin: \n");
```

```

scanf("%c",&a);

if(a=='a'){

printf("Daxil etdiyiniz hərf a hərfidir və əlifbada birinci hərfdir.");

}

else if(a=='b'){

printf("Daxil etdiyiniz hərf b hərfidir və əlifbada ikinci hərfdir.");

}

else if(a=='c'){

printf("Daxil etdiyiniz hərf c hərfidir və əlifbada üçüncü hərfdir.");

}

else{

printf("Daxil etdiyiniz hərf a,b və c-dən fərqlidir.");

}

getchar();

return 0;

}

```

Gördüyümüz kimi, istifadəçidən bir hərf daxil etməsini tələb etdik və daxil olunan hərfi şərtlərlə yoxlayaraq müxtəlif əməliyyatlar yazdıq. Burada aşağıdakı hissəyə diqqət etməliyik:

if(a=='a'){

İstifadə etdiyimiz char tipli dəyişəni şərtlə yoxlayarkən, simvolu tək dırnaqlar arasında yazmağımız mütləqdir. Cüt dırnaq yazdıqda proqram işləməyəcəkdir.

İndi isə, müxtəlif dəyişən tiplərindən istifadə edərək nisbətən daha maraqlı bir proqram yazaq. İstifadəçidən iki ədəd və bir simvol daxil etməsini tələb edək. Daxil etdiyi simvol «+» olsa bu iki ədədi toplayaq, «-» olsa çıxmaq, «*» olsa bir-birinə vuraq, «/» olsa isə bölək. Edəcəyimiz əməliyyatlar sadədir, sadəcə iki ədədi

və bir simvolu istifadəçidən qəbul edəcəyik və bu simvolu şərtlə yoxlayaraq müxtəlif əməliyyatlar aparacağıq. İndi isə, gəlin proqramı yazaq:

```
#include <stdio.h>

int main(){

float a,b;

char simvol;

printf("Birinci ədədi daxil edin: \n");

scanf("%f", &a);

printf("İkinci ədədi daxil edin: \n");

scanf("%f", &b);

printf("Simvolu daxil edin(+,-,*,/): \n");

scanf("%c", &simvol);

if(simvol=='+'){

printf("%f",a+b);

}

else if(simvol=='-'){

printf("%f",a-b);

}

else if(simvol=='*'){

printf("%f",a*b);

}

else if(simvol=='/'){

printf("%f",a/b);

}

else{
```

```
printf("Daxil etdiyiniz simvol düzgün deyil.");  
  
}  
  
getchar();  
  
return 0;  
  
}
```

Gördüyümüz kimi, ilk öncə 3 dəyişən elan etdik. Növbəti olaraq iki ədədi və xarakterimizi qəbul etdik. Növbəti olaraq isə xarakterin hansı simvol olduğunu yoxlayaraq ona uyğun əməliyyat apardıq. Fikrimcə bu daha maraqlı oldu :)

Qeyd: Əgər yuxarıdakı proqramın istifadəsi zamanı istədiyimiz nəticəni almasanız, simvolu ikinci ədədin sağında daxil edin. Yəni, ikinci ədədi daxil etdikdən sonra, enter düyməsinə basmadan əvvəl simvolu da daxil edin. Bu zaman gözlədiyimiz nəticəni alacaqsınız.

Sabitlər

Dəyişənlərdən əlavə proqramın gedişatında sabit qiymətləri qeyd etmək üçün sabitlər mövcuddur. Sabitlər const açar sözü ilə təyin olunur. Məsələn:

```
#include <stdio.h>  
  
int main(){  
  
const x=5;  
  
printf("%d",x);  
  
return 0;  
  
}
```

Burada bir sabit elan edib onu çap edirik. Yazdığımız nümunə sadə olduğu üçün, sabitlərdən niyə istifadə etməli olduğumuzu düşünə bilərik. Ancaq bəzən həqiqətən də sabitlərə ehtiyac duyulur. Məsələn, proqram ərzində 100 dəfə e

ədədinin qiymətini çap edəcəyiksə, hər dəfə e-nin qiymətini yazmaq yerinə, bir dəfə e-i sabit olaraq qeyd edib digər yerlərdə istifadə edə bilərik. Məsələn:

```
#include <stdio.h>

int main(){

const double e=2.718281828;

printf("%f",e);

return 0;

}
```

Burada e ədədinin qiyməti double tipli olduğu üçün const açar sözündən sonra double sözünü də yazırıq.

Çoxluqlar

Çoxluq anlayışı yəqin ki, riyaziyyatdan sizə tanış olar. Məsələn, bir A çoxluğu göstərək:

$A=\{4,3,5\}$

Gördüyümüz kimi, bir A çoxluğudur və içərisində üç element mövcuddur. Bu elementlər 4, 3 və 5-dir. İlk olaraq riyaziyyatda göstərsək də əslində, çoxluqları həyatın hər bir nöqtəsində görə bilərik. Riyaziyyatda çoxluğun elementləri yalnız ədədlər olaraq göstərilərsə də, praktikada ədədlərin yerini real elementlər, zərrəciklər canlılar ala bilir. Məsələn, bütün insanlığı 7 milyard elementli olan bir çoxluq olaraq görə bilərik. Bütün Azərbaycanı 10 milyonluq bir çoxluq olaraq görə bilərik. Günəş sistemini içərisində 9 elementi, yəni 9 planeti olan bir çoxluq olaraq görə bilərik. Kainatı içərisində saysız-hesabsız planet, meteorit, zərrəcik, element olan çoxluq olaraq görə bilərik. Gördüyümüz kimi, çoxluqlar hər yerdə var. Hər yerdə olduğu kimi, proqramlaşdırmada da çoxluq anlayışı var və istifadəsi də çox genişdir. Proqramlaşdırmadakı çoxluqlar adətən ədədlərdən, simvollarıdan, yazılardan ibarət ola bilər.

İndi isə, real kod üzərində bir çoxluğun elan olunma qaydasını və istifadəsini göstərək.

```
#include <stdio.h>

int main(){

int A[3];

A[0]=5;

printf("%d",A[0]);

getchar();

return 0;

}
```

Yazdığımız kodu izah etməyə çalışaq. İlk olaraq A çoxluğunu elan etdik. A çoxluğunda 3 element var. Yox, yox, burada bir fərqlilik var. Həqiqətəndə mi A çoxluğunda 3 element var? Cavabı: A çoxluğunda 3 element yox 4 element var. Səbəbi isə, proqramlaşdırmada saymağa 1-dən yox, 0-dan başlanılmasıdır. İlk öncə 0-cı element sonra 1-ci element, sonra 2-ci element, sonra isə 3-cü element gəlir, nəticədə çoxluqda 4 element olmuş olur. Yəni real çoxluqdakı 1-ci element proqramlaşdırmada 0-cı element olaraq göstərilir.

Növbəti olaraq A[0]=5; yazaraq A çoxluğunun 0-cı elementini 5-ə bərabər etdik və sonra onu çap etdik. Çoxluq olsa da, çoxluğun tipi int olduğu üçün, çoxluğun 0-cı elementi printf() funksiyasında çap edilərkən «%d» hissəsindən istifadə olunur. A çoxluğundakı 1-ci, 2-ci və 3-cü elementlərə heç bir ədəd vermədiyimiz üçün çoxluğun bu elementləri boş qalmış olur.

Bu sadə nümunədə çoxluq istifadə etməsək belə eyni əməliyyatı yerinə yetirə bilərdik. Ancaq, növbəti mövzularda çoxluğun nə qədər mühüm əhəmiyyət kəsb etdiyini görəcəyik.

İndi isə simvol çoxluğuna nəzər salaq. Simvol çoxluğu nədir? Cavab: söz, cümlə, ifadə. Deməli biz simvol çoxluğu elan edərək istifadəçidən tək simvolları yox, bütöv bir sözü də qəbul edə bilərik. Biraz daha ətraflı izah edək. Char tipli bir dəyişən elan etdikdə, bu dəyişənə yalnız bir simvol mənimsədə bilirik, yəni scanf funksiyas ilə bu dəyişən üçün istifadəçidən yalnız bir simvol qəbul edə bilirik.

Ancaq çoxluq elan etdikdə, simvol çoxluğu yarandığı üçün, istifadəçi bütöv bir sözü daxil edə bilir. Real proqram üzərində göstərək.

```
#include <stdio.h>

int main(){

char A[10];

scanf("%s",A);

printf("\n Yazdığınız söz: %s",A);

getchar();

return 0;

}
```

Gördüyümüz kimi, char tipli A çoxluğunua elan etdik, növbəti olaraq A çoxluğunu istifadəçidən qəbul etdik və daxil etdiyi sözü printf() funksiyası vasitəsilə çap etdik. Diqqət edək ki, scanf() və printf() funksiyaları daxilində A çoxluğunun sıra nömrəsi olmadan hər-hansı bir elementini yox, birbaşa özünü göstərdik. Simvollar çoxluğu bir ifadəni əmələ gətirdiyi üçün bu şəkildə yazmağımız düzgündür. Ancaq digər tiplərdə, çoxluğun sıra nömrəsini göstərməyimiz mütləqdir. Eyni zamanda, scanf() funksiyasında çoxluğun istifadəsinə bir daha diqqət edək:

```
scanf("%s",A);
```

Gördüyümüz kimi, burada çoxluq olduğu üçün «&A» yox, sadəcə A yazdıq.

İndi isə, gəlin istifadəçidən bir şifrə daxil etməsini tələb edək və şifrə doğru olarsa, doğru daxil etdiyini deyək. Şifrəmiz isə «salam» olsun.

```
#include <stdio.h>

int main(){

char A[10];

scanf("%s",A);

printf("%s",A);

if(A[0]=='s' && A[1]=='a' && A[2]=='l' && A[3]=='a' && A[4]=='m'){
```

```

printf("\nDoğru daxil etdiniz.");

}

else{

printf("\nYalnış daxil etdiniz.");

}

getchar();

return 0;

}

```

Gördüyümüz kimi, istifadəçidən çoxluğu qəbul edirik və çoxluğun simvollarını şərt operatoru vasitəsilə bir-bir yoxlayaraq yazılan sözün «salam» olub-olmadığını yoxlayırıq.

Müasir proqramlaşdırma dillərində, hazır yazı tipi(String) mövcuddur ki, bu tip vasitəsilə simvol çoxluğu elan etmədən birbaşa ifadəni daxil etmək mümkün olur. Bunu C++ bölməsinə çatanda daha ətraflı göstərəcəyik.

İndi isə, kiçik bir tərcümə proqramı yazmağa nə deyirsiniz? Fikrimcə maraqlı olar :)

Edəcəyimiz şey sadədir, bir simvol çoxluğu elan edəcəyik, sonra onu istifadəçidən qəbul edəcəyik və onun hansısa bir söz olub-olmadığını yoxlayacağıq. Biz bu nümunədə qısa olsun deyə sadəcə 3 sözü üçün («hello», «apple», «friend») tərcüməni yazacağıq, amma, siz istəyinizə uyğun bu sözlərin sayını artırma bilərsiniz. Hətta proqram dostlarınızla paylaşa, özünüz üçün də istifadə edə bilərsiniz.

```

#include <stdio.h>

int main(){

char A[20];

scanf("%s",A);

printf("%s",A);

if(A[0]=='h' && A[1]=='e' && A[2]=='l' && A[3]=='l' && A[4]=='o'){

printf("\nSözün tərcüməsi: Salam \n");

```

```

}

else if(A[0]=='a' && A[1]=='p' && A[2]=='p' && A[3]=='l' && A[4]=='e'){

printf("\nSözün tərcüməsi: Alma \n");

}

else if(A[0]=='f' && A[1]=='r' && A[2]=='i' && A[3]=='e' && A[4]=='n' &&
A[5]=='d'){

printf("\nSözün tərcüməsi: Dost \n");

}

else {

printf("\nDaxil etdiyiniz söz tapılmadı. \n");

}

getchar();

return 0;

}

```

Sözləri daxil edərkən kiçik hərflə daxil etdiyimizə xüsusi diqqət etməliyik. Burada böyük hərflərlə kiçik hərflər fərqlənir. İndi isə proqramdakı sözlərin sayını artırın və proqramı dostlarınızla da paylaşın :)

Dövrələr

Bir otaqda 100 ədəd qutu olduğunu və bu qutuladan neçəsinin boş, neçəsinin dolu olduğunu öyrənmək istədiyimizi fərz edək. Bunun üçün qutuları bir-bir açıb qutuların dolu olub-olmadığına baxmalıyıq. Buna oxşar əməliyyatları proqramlaşdırmada da yerinə yetirərkən eyni üsuldan istifadə etməliyik. Yazacağımız xəyali proqram qutuları bir-bir açıb hansının dolu, hansının boş olmasını yoxlamalıdır. 100 dəfə qutuları yoxlamaq isə bir xeyli vaxtımızı alacaq. Proqramlaşdırmada sırf bu cür ardıcıl əməliyyatları qısa şəkildə yerinə yetirmək üçün, eyni bir əməliyyatı bir neçə dəfə təkrarlamaq üçün bir sıra operatorlar mövcuddur. Bu operatorlardan biri də **while** operatorudur. Bu operator, bir şərt ödəndiyi müddətcə bir əməliyyatı yerinə yetirmək üçündür. Operatorun istifadə qaydası if operatoruna oxşardır. Əvvəlcə operatorun adı yazılır, sonra mötərizə daxilində şərt, sonra isə fiqurlu mötərizələr arasında əməliyyatlar yazılır. Nümunə bir proqramda bu operatorndan istifadə edək.

```
#include <stdio.h>
```

```
int main(){
```

```
int i=0;
```

```
while(i<10){
```

```
printf("While\n");
```

```
}
```

```
getchar();
```

```
return 0;
```

```
}
```

Proqramı işlətdikdən sonra, alt-alta daima «While» sözünün çap olunduğunu görəcəyik. Çünki, operatorda şərt olaraq i dəyişəninin 10-dan kiçik olduğu halları yazmışıq. Bu dəyişən isə 0-a bərabərdir və həmişə 10-dan kiçikdir. Buna görə də operatorun daxilindəki əməliyyatlar həmişə, sonsuzluğa qədər işləməyə davam edəcək, yəni, sonsuz sayda alt-alta «While» yazısı çap olunacaq.

«While» operatorunda ilk olaraq yazdığımız şərt yoxlanılır. Burada yazdığımız şərt i dəyişəninin 10-dan kiçik olmasıdır. Şərt ödəndiyi üçün ilk olaraq

əməliyyatlar yerinə yetirilir. Əməliyyatlar bir dəfə yerinə yetirildikdən sonra, operator yenidən əvvələ qayıdaraq şərti bir daha yoxlayır. Şərt doğru olarsa yenidən əməliyyatlar aparılır. Bu proses daimi olaraq davam edir. Yuxarıdakı nümunədə də şərt həmişə ödənəcəyinə görə, əməliyyatlar həmişə yerinə yetirilir.

İndi isə əməliyyatları sonsuz sayda yox, 5 dəfə yerinə yetirməyə çalışaq. Bunun üçün yuxarıdakı kodda kiçik bir dəyişiklik edəcəyik.

```
#include <stdio.h>
```

```
int main(){
```

```
int i=0;
```

```
while(i<5){
```

```
printf("While\n");
```

```
i++;
```

```
}
```

```
getchar();
```

```
return 0;
```

```
}
```

Burada ilk olaraq i dəyişənin qiyməti 0-dır. İlk şərtə dəyişənin qiymətinin 5-dən kiçik olub-olmaması yoxlanılır. Həqiqətən də 0 rəqəmi 5-dən kiçikdir. Şərt ödəndiyinə görə əməliyyatlar yerinə yetirilməyə başlanır. İlk əməliyyatımız sözün çap olunmasıdır. Sonrakı əməliyyatımız isə dəyişənin qiymətinin 1 vahid artmasıdır. İlk dövrün sonunda dəyişənin qiyməti 1 vahid artaraq 1 olur.

Əməliyyatlar bir dəfə yerinə yetirildikdən sonra operator başa qayıdır və şərti yenidən yoxlayır. Bu dəfə i dəyişənin qiyməti 1-ə bərabərdir. Həqiqətən də 1 rəqəmi 5-dən kiçik olduğu üçün şərt ödənilir və operator əməliyyatları yerinə yetirməyə başlayır, eyni zamanda əməliyyatların sonunda i++ ifadəsi ilə dəyişənin qiymətini 1 vahid artıraraq 2 edir.

Artıq əməliyyatlar iki dəfə yerinə yetirildi. Növbəti dəfə yenə operator əvvələ qayıdaraq i dəyişənin qiymətinin 5-dən kiçik olub-olmamasını yoxlayır. Dəyişənin hal-hazırkı qiyməti 5-dən kiçik olduğu üçün, yəni şərt ödəndiyi üçün,

əməliyyatlar yerinə yetirilməyə başlayır və eyni zamanda *i* dəyişənin qiyməti 1 vahid artaraq 3 olur.

Üçüncü dəfə əməliyyatlar yerinə yetirildikdən sonra, program yenidən əvvələ qayıdır və *i* dəyişənin qiymətinin 5-dən kiçik olub-olmamasını yoxlayır. Bu dəfə də *i* dəyişənin qiyməti, yəni 3, 5-dən kiçik olduğu üçün şərt ödənilir və əməliyyatlar yerinə yetirilməyə başlayır, eyni zamanda *i* dəyişənin qiyməti 1 vahid artaraq 4 olur.

Dördüncü dəfə də əməliyyatlar yerinə yetirildikdən sonra, operator yenidən əvvələ qayıdır və *i* dəyişənin 5-dən kiçik olub-olmamasını yoxlayır. Bu dəfə də *i* dəyişənin qiyməti, yəni 4, 5-dən kiçik olduğu üçün şərt ödənilir və əməliyyatlar yerinə yetirilməyə başlayır, eyni zamanda *i++* yazıldığı üçün, *i* dəyişənin qiyməti 1 vahid artaraq 5 olur.

Artıq əməliyyatlar 5 dəfə yerinə yetirilib və *i* dəyişənin qiyməti 5-ə bərabər olub. 5-ci dəfə əməliyyatlar yerinə yetirildikdən sonra, operator yeindən əvvələ qayıdaraq *i* dəyişənin qiymətinin 5-dən kiçik olub olmadığını yoxlayır. Bu dəfə *i* dəyişənin qiyməti 5-dir və 5-dən kiçik deyil, 4 olsa idi doğru olacaqdı. Beləliklə şərt ödənilir, şərt ödənilmədiyinə görə, görə operator bu dəfə əməliyyatları yerinə yetirmir və operatorun işi sona çatmış olur. Bundan sonra artıq operatorun sonrakı əməliyyatlar yerinə yetirilməyə başlayır.

Bu operatora aid bir neçə nümunə daha yazaraq operatorun işini daha yaxşı anlamağa çalışaq. Gəlin 1-dən 10-a qədər ədədləri alt-alta çap edən bir program yazaq.

```
#include <stdio.h>
```

```
int main(){
```

```
int i=1;
```

```
while(i<11){
```

```
printf("%d\n",i);
```

```
i++;
```

```
}
```

```
getchar();
```

```
return 0;

}
```

Dəyişənimizin qiyməti əvvəlcə 1 olaraq təyin olunmuşdur. İlk olaraq operatorda dəyişənin qiymətinin 11-dən kiçik-bərabər olub-olmaması yoxlanılır. Şərt doğru olduğu üçün əməliyyatlar yerinə yetirilməyə başlayır. Əməliyyatlarımız i dəyişənin çap olunmasından və sonda bir vahid artırılmasından ibarətdir. İlk olaraq i dəyişənin qiyməti 1 çap olunur və bir vahid də artaraq 2 olur. Sonra yenidən operator başa qaydır və şərt yoxlanılır. Doğru olduğu üçün əməliyyatlar yerinə yetirilir, i-nin qiyməti çap olunur və bir vahid artaraq 3 olur. Bu proses i-nin qiyməti 11 olana qədər davam edir. Dəyişənin qiyməti 11 olduqda artıq şərt ödənmədiyinə görə əməliyyatlar başlamır və operator öz işini bitirmiş olur.

İndi isə, 1-dən 5-ə qədər ədədlərin cəmini tapan bir proqram yazaq.

```
#include <stdio.h>

int main(){

int i=0;

int cem=0;

while(i<5){

i++;

cem=cem+i;

}

printf("%d",cem);

getchar();

return 0;

}
```

Burada hər dövrdə i-nin qiyməti 1 vahid artmış olur və i-nin yeni qiyməti cem dəyişənin üzərinə gəlinir. Cem dəyişənin ilk qiyməti 0-dır. İlk dövrün sonunda 1 olur. Növbəti dövrdə 3 olur, növbəti dövrdə 6, növbəti dövrdə 10, sonuncu dövrdə isə 15 olur və operator öz işini bitirmiş olur.

İndi isə, 1-dən 5-ə qədər ədədlərin hasilini tapan bir proqram yazaq.

```
#include <stdio.h>

int main(){

int i=0;

int hasil=1;

while(i<5){

i++;

hasil=hasil*i;

}

printf("%d",hasil);

getchar();

return 0;

}
```

Burada hasil dəyişəninin ilk qiyməti 1 olaraq təyin olunmuşdur. İlk dövrdə hasil dəyişəni 1-ə vurulur, ikinci dövrdə 2-ə vurularaq 2 olur, üçüncü dövrdə 3-ə vurularaq 6 olur, dördüncü dövrdə 4-ə vurularaq 24 olur, beşinci və sonuncu dövrdə isə 5-ə vurularaq 120 olur və operator öz işini tamamlayır.

Burada diqqət etməli olduğumuz əsas məqam, cəmin tapılarkən ilk qiymətin 0, hasilin tapılarkən ilk qiymətin 1 olaraq təyin olunmasıdır. Cəm tapılarkən ilk qiymət 0 olaraq təyin olunur, çünki, növbəti ədəd 0-ın üzərinə gəlinəcək və cəm alınacaq. Hasilə isə 1 olur, çünki növbəti ədəd 1-ə vurulacaq. Əgər hasildə də ilk qiymət 0 olsa, onda növbəti ədəd 0-a vurular və yeni qiymət də 0 olardə. Nəticədə isə dəyişənin son qiyməti 0 olaraq qalardı və istədiyimiz nəticəni ala bilməzdik.

İndi isə, gəlin 1 ilə 100 arasında olan cüt ədədləri çap edək. Əvvəlcə yalnız 1 ədədin cüt olub-olmadığını yoxlamaq üçün lazım olan koda baxaq:

```
#include <stdio.h>

int main(){

int eded=51;
```



```

if(eded%2==0) printf("Cut");

else printf("Tek");

getchar();

return 0;

}

```

Burada əsas if hissəsinə diqqət edək. «`eded%2`» ifadəsi, dəyişənin 2-ə bölünməsindən alınan qalıqı göstərir. Məsələn, ədədimiz 51-dirsə, 2-ə bölünməsindən alınan qalıq 1 olur. Çünki, 51 tək ədəddir. Cüt ədədlərdə isə, qalıq 0 olur. Bundan istifadə edərək ədədin cüt yoxsa tək olmasını ayırd edə bilirik. Yəni, əgər ədədin 2-ə bölünməsindən alınan qalıq 0-dırsa ədəd cütdür, 1-dirsə ədəd təkdir. İndi isə bunu dövr daxxilində yazaraq 1-dən 100-ə qədər olan cüt ədədləri tapaq:

```

#include <stdio.h>

int main(){

int i=1;

while(i<=100){

if(i%2==0) printf("%d\n",i);

i++;

}

getchar();

return 0;

}

```

Beləliklə, 1 ilə 100 arasında olan cüt ədədləri tapmış oluruq. Əgər şərt daxilində «`i%3==0`» yazsaq, 1 ilə 10 arasında 3-ə qalıqsız bölünən ədədləri, «`i%5==0`» yazsaq 1 ilə 100 arasında 5-ə qalıqsız bölünən ədədləri tapmış olarıq.

System funskiyası

System funksiyası Windowsda Cmd, Linuxda Terminal əmrlərini C dilində istifadə etmək üçündür. Cmd pəncərəsini açmaq üçün Windows+r düymələrini bir yerdə basıb, açılan kiçik pəncərədə “cmd” yazıb enter düyməsinə sıxmalıyıq. Düyməyə basdıqdan sonra cmd pəncərəsi açılacaq. Bura yazacağımız ifadələrin eynisini system() funksiyası daxilinə yazaraq eyni əməliyyatları yerinə yetirə bilərik. Məsələn, “cls” parametri ilə funksiya ekranı təmizləyə bilir, yəni funksiya işlədilməzdən əvvəl çap edilmiş bütün yazıları silinir. Bu əmri cmd pəncərəsində yazaraq yoxlaya bilərsiniz, nəticədə, ekrandakı yazılar silincəkdir. Eyni zamanda, ifadəni system() funksiyasının daxilində də istifadə edə bilərik. Nümunəyə baxaq:

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main(){
```

```
printf("Old");
```

```
system("cls");
```

```
printf("New");
```

```
return 0;
```

```
}
```

Gördüyümüz kimi, proqrama “windows.h” kitabxanası əlavə olunmuşdur. Proqramın nəticəsində ekranda yalnız “New” sözü çap olunacaq. Çünki, system("cls") funksiyası ilə ondan əvvəl çap edilmiş yazılar silinmiş olur.

System funksiyasının parametrini dəyişməklə arxaplan rəngini və yazıların rəngini də dəyişmək mümkündür. Nümunə ilə bunu göstərək.

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main(){
```

```
system("color 2");
```

```
printf("Word");
```

```
return 0;  
  
}
```

Nəticədə yazıların rəngi dəyişərək yaşıl rəngdə olmuş olur. Gördüyümüz kimi, `system()` funksiyasının içində “color” yazısını və 2 rəqəmini yazdıq. “Color” adından da göründüyü kimi rəngi bildirir. 2 rəqəmi isə yaşıl rəngi bildirir. 2 rəqəmini dəyişməklə yazıların rəngini də dəyişə bilərik. Rəng kodları aşağıdakılardır:

0=Qara; 1=Göy; 2=Yaşıl; 3=Su rəngi; 4=Qırmızı; 5=Bənövşəyi; 6=Sarı; 7=Ağ; 8=Boz;
9=Açıq göy; A=Açıq yaşıl; B=Açıq su rəngi; C=Açıq qırmızı; D=Açıq bənövşəyi;
E=Açıq sarı; F=Parlaq ağ;

Yazının rəngi ilə bərabər fon rəngini də dəyişmək mümkündür. Bunun üçün sadəcə, rəngi bildirən ədədi iki dəfə yazmalıyıq. Məsələn, yaşıl fonda göy rəngli yazı üçün “color 12” yazmalıyıq.

```
#include <windows.h>  
  
#include <stdio.h>  
  
int main(){  
  
system("color 21");  
  
printf("Word");  
  
return 0;  
  
}
```

Burada birinci rəqəm fon rəngini, ikinci rəqəm isə yazının rəngini bildirir. Açıq yaşıl fonda qara rəngdə yazı yazmaq üçün isə sadəcə `system("color A0")`; yazmağımız kifayətdir. Burada A açıq yaşıl fon rəngini, 0 isə qara yazı rəngini bildirir.

“Windows.h” kitabxanası

İstifadə edəcəyimiz kitabxanalardan biri də, “windows.h” kitabxanasıdır. Kitabxananın müxtəlif funksiyaları ilə müxtəlif əməliyyatlar yerinə yetirmək mümkündür. Kitabxananın funksiyalarından biri Sleep funksiyasıdır. Sleep funksiyası, bir əməliyyatı müəyyən qədər gecikdirmək üçün istifadə olunur. Məsələn, biz istəyirik ki, bir yazı çap olunsun, ondan 3 saniyə sonra isə digər yazı çap olunsun. Bunun üçün sleep funksiyasından istifadə edə bilərik.

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main(){
```

```
printf("Word1");
```

```
Sleep(3000);
```

```
printf("\nWord2");
```

```
return 0;
```

```
}
```

Burada diqqət edəcəyimiz əsas şeylər, ilk olaraq sleep sözünün ilk hərflə böyük yazılması, parametrin isə saniyə yox, millisaniyə ilə göstərilməsidir. Buna görə də, 3 saniyə üçün parametrlə olaraq 3000 daxil etmişik. Proqramı işə saldıqdan sonra nəticə olaraq, ilk öncə “Word1” yazısı, ondan 3 saniyə sonra isə “Word2” yazısı çap olunacaq.

Bu funksiyalardan və while operatorundan istifadə edərək bir timer hazırlamağımız mümkündür. Timer 1-dən başlayaraq yuxarıya doğru saymağa başlayacaq.

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main(){
```

```
int i=1;
```

```
while(1==1){
```

```

printf("%d",i);

Sleep(1000);

i++;

system("cls");

}

return 0;

}

```

İlk olaraq `i` dəyişəni elan edirik və ona 1 qiymətini mənimsədirik. Növbəti olaraq isə, ədədin daima artmasının davam etməsi üçün `while` operatoru ilə sonsuz dövrə yaradırıq. `1==1` şərti həmişə ödəndiyi üçün sonsuz dövrə yaranmış olur. Operator daxilində ilk əməliyyatımız `i` dəyişəninin qiymətinin çap olunmasıdır. İlk olaraq `i` dəyişəni çap olunur və 1 qiyməti yazılır. Növbəti olaraq `sleep` funksiyası vasitəsilə 1 saniyə gözlənilir. 1 saniyə keçdikdən sonra `i` dəyişəninin qiyməti 1 vahid artırılır. Sonra isə əvvəl yazılmış qiymətin ekrandan silinməsi üçün `system("cls");` yazılır və ekran təmizlənir. Sonra isə operator yenidən başa qaydır, yeni qiyməti ekrana yazır və bu əməliyyatlar sonsuz olaraq davam edir. Əgər `system("cls");` yazılmasa idi qiymətlər yan-yanə yazılacaqdı və timer effekti alınmayacaqdı. Məsələn: 123456789...

İstifadə edəcəyimiz funksiyalardan biri də `Beep()` funksiyasıdır. `Beep()` funksiyası signal vermək üçün istifadə olunur. Funksiya iki parametr alır, birinci parametr səsin tezliyi, ikinci parametr isə səsin davamətmə müddətidir. Səsin davamətmə müddəti millisaniyə ilə göstərilir.

```

#include <windows.h>

#include <stdio.h>

int main(){

    Beep(1000, 500);

    return 0;

}

```

Nəticədə, tezliyi 1000 olan signal 500 millisaniyə müddətində səslənəcək. Programı işlədərkən qulaqcıqdan istifadə etməyi unutmayın 😊

Bu funksiyanı yuxarıda yazdığımız taymerə tətbiq edərək, hər bir ədəd artarkən signal verilməsini təmin edə bilərik.

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main(){
```

```
int i=1;
```

```
while(1==1){
```

```
printf("%d",i);
```

```
Sleep(500);
```

```
i++;
```

```
system("cls");
```

```
Beep(1000, 500);
```

```
}
```

```
return 0;
```

```
}
```

Burada Beep() funksiyasının özünün də işləməsi üçün 500 millisaniyə tələb olunduğuna görə, sleep funksiyasında da parametri dəyişərək 500 edirik. Çünki Beep() funksiyasında da 500 daxil etdiyimiz üçün səs 500 millisaniyə davam edəcək və signal səsləndirilərkən gözləmə olacaq. 500 millisaniyə sleep() funksiyası ilə, 500 millisaniyə isə Beep() funksiyası ilə gözləmə olacağı üçün nəticədə bir saniyə gözləmə baş verəcək.

İndi isə, fon rəngini bir saniyədən bir dəyişən işıqfor programı yazaq:

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main(){
```

```

char reng='r';

while(1==1){

if(reng=='r'){

system("color 40");

reng='g';

}

else if(reng=='g'){

system("color 60");

reng='b';

}

else if(reng=='b'){

system("color 20");

reng='r';

}

Sleep(1000);

}

return 0;

}

```

Nəticədə, pəncərənin fon rəngi bir saniyədən bir qırmızı, sarı və yaşıl olaraq dəyişəcək. Əgər color dəyişəninin qiyməti r(red) olarsa fon rəngi qırmızı, g(green) olarsa yaşıl, b(blue) olarsa göy olacaq.

Beep() funksiyasından istifadə edərək kiçik sadə melodiylar da yaratmaq mümkündür. Bunun üçün, funksiyadan bir neçə dəfə istifadə etmək və istifadə edərkən də düzgün tezliyi və düzgün zaman müddətini seçmək lazımdır. Məsələn, aşağıda mario oyunu üçün sadə bir melodiya kodu göstərilmişdir.

```
#include <windows.h>
```

```
int main(){  
    Beep (330,100);Sleep(100);  
    Beep (330,100);Sleep(300);  
    Beep (330,100);Sleep(300);  
    Beep (262,100);Sleep(100);  
    Beep (330,100);Sleep(300);  
    Beep (392,100);Sleep(700);  
    Beep (196,100);Sleep(700);  
    Beep (196,100);Sleep(125);  
    Beep (262,100);Sleep(125);  
    Beep (330,100);Sleep(125);  
    Beep (392,100);Sleep(125);  
    Beep (523,100);Sleep(125);  
    Beep (660,100);Sleep(125);  
    Beep (784,100);Sleep(575);  
    Beep (660,100);Sleep(575);  
    Beep (207,100);Sleep(125);  
    Beep (262,100);Sleep(125);  
    Beep (311,100);Sleep(125);  
    Beep (415,100);Sleep(125);  
    Beep (523,100);Sleep(125);  
    Beep (622,100);Sleep(125);  
    Beep (830,100);Sleep(575);  
    Beep (622,100);Sleep(575);  
}
```



```

Beep (233,100);Sleep(125);
Beep (294,100);Sleep(125);
Beep (349,100);Sleep(125);
Beep (466,100);Sleep(125);
Beep (587,100);Sleep(125);
Beep (698,100);Sleep(125);
Beep (932,100);Sleep(575);
Beep (932,100);Sleep(125);
Beep (932,100);Sleep(125);
Beep (932,100);Sleep(125);
Beep (1046,675);
return 0;
}

```

Nəticədə kiçik, sadə bir melodiya səslənəcək. Digər melodiyları yaratmaq üçün proqramlaşdırma biliklərindən əlavə musiqi biliklərinə də ehtiyac duyulur. Musiqidə olan 7 nota uyğun tezliklər və zaman müddətləri aşağıdakı şəkildədir:

```

Do — Tezlik: 523; Zaman: 191ms — 382ms;
Re — Tezlik: 587; Zaman: 170ms — 340ms;
Mi — Tezlik: 659; Zaman: 152ms — 304ms;
Fa — Tezlik: 698; Zaman: 143ms — 286ms;
Sol — Tezlik: 740; Zaman: 135ms — 270ms;
Lya — Tezlik: 880; Zaman: 114ms — 228ms
Si — Tezlik: 988; Zaman: 110ms - 220ms

```

For operatoru

Yuxarıdakı nümunələrdə dövr yaratmaq üçün «while» operatorundan istifadə etdik. Ancaq, dövr yaratmaq üçün «while» operatorundan əlavə «for» operatorundan istifadə olunur. «While» operatorunda dövr sayı əvvəlcədən bilinməsə də dövrü başlatmaq olurdu və dövr sayı sonsuz da ola bilər. For operatoru isə əsasən dövr sayı məlum olduqda istifadə olunur. Operatorun istifadə qaydasına aid bir nümunə göstərək:

```
#include <stdio.h>

int main(){

int i;

for(i=1;i<=5;i++){

printf("%d\n",i);

}

return 0;

}
```

While operatorunda i dəyişəninin artımını yalnız operatorun əməliyyatların daxilində göstərə bilirdik. Ancaq, for operatorunda, əməliyyatlar başlamazdan öncə i dəyişəninin ilk qiymətini təyin edirik. İlk parametrdə i dəyişəninin ilk qiyməti təyin olunur. İkinci parametrdə şərt yazılır, şərtimiz dəyişənin 5-dən kiçik-bərabər olmasıdır. Üçüncü parametr isə i-nin artımıdır. Operatorun işləmə qaydası da while operatorunun işləmə qaydasına oxşardır. İlk olaraq i-nin ilk qiyməti təyin olunduqdan sonra i-nin qiymətinin 5-dən kiçik-bərabər olub-olmaması yoxlanılır. Şərt ödənirsə əməliyyatlar başlayır, sonda isə i-nin qiyməti 1 vahid artırılır və yenidən əvvələ qaydılır. i-nin qiyməti 1 vahid artdıqdan sonra şərt yenidən yoxlanılır. Şərt ödənərsə proses eyni qaydada yerinə yetirilir. Bu proses şərt ödənməyə qədər davam edir. Dəyişənin qiyməti 5-ə çatdıqda əməliyyatlar şərt ödəndiyinə görə yerinə yetirilir və dəyişənin qiyməti bir vahid artırılaraq altı olur. Yenidən əvvələ qaydılır və şərt yoxlanılır. Şərt ödənilmədiyinə görə əməliyyatlar yerinə yetirilmir və operator öz işini bitirir.

While operatoru ilə yazdığımız bir çox nümunəni eynilə for operatoru ilə də yazmağımız mümkündür. Məsələn, 1-dən 100-ə qədər olan cüt ədədlərin

tapılması üçün lazım olan proqramı for operatorundan istifadə edərək də yazı bilərik.

```
#include <stdio.h>

int main(){

int i;

for(i=1;i<=100;i++){

if(i%2==0) printf("%d\n",i);

}

return 0;

}
```

Nəticədən 1-dən 100-ə qədər olan cüt ədədlər alt-alta çap olunmuş olacaq. Dövr sayı əvvəlcədən məlum olan dövrlər üçün for operatorunun istifadə olunması daha məqsəduyğundur.

For operatoru ilə başqa bir nümunə də yazıq. Məsələn, ədədin faktorialını hesablamaq üçün for operatorundan istifadə edə bilərik. Faktorialın nə olduğunu xatırlayıq, faktorial 1-dən təyin etdiyimiz ədədə qədər olan ədədlərin hasilidir. Məsələn, 4 faktorial $4*3*2*1$ deməkdir. Deməli bir ədədin faktorialını tapmaq üçün, 1-dən başlayaraq həmin ədədə qədər olan ədədləri bir-birinə vurmalıyıq. Aşağıdakı nümunədə proqramla 5 faktorialı hesablayıq.

```
#include <stdio.h>

int main(){

int faktorial=1;

int i;

for(i=1;i<=5;i++){

faktorial=faktorial*i;

}

printf("%d\n",faktorial);

}
```

```
return 0;  
  
}
```

Yazdığımız bu proqramla 5 faktorialı hesablamış oluruq. İlk olaraq bir faktorial dəyişəni elan edirik və onun qiymətini 1-ə bərabər edirik. Sonra isə for operatoru ilə i dəyişənini 1-dən 5-ə qədər sıralayırıq və faktorial dəyişənini i-nin aldığı hər bir qiymətə vururuq, yəni faktorial dəyişənini 1-dən 5-ə qədər bütün ədədlərə vururuq və nəticədə 5 faktorialı tapmış oluruq.

Do While Operatoru

Dövr operatorlarından biri də «do while» operatorudur. Bu operator «while» operatoruna çox oxşardır. Ancaq əsas bir fərqi mövcuddur. «while» operatorunda şərt ödənmədiyi halda əməliyyatlar yerinə yetirilmirdi. «do while» operatorunda isə şərt ödənməsə belə əməliyyatlar ən azı 1 dəfə yerinə yetirilir və operator öz işini bitirir. Bəzi yerlərdə şərt ödənməsə belə əməliyyatların ən azı bir dəfə yerinə yetirilməsinə ehtiyac duyularsa, «do while» operatorundan istifadə oluna bilər. Operatorun istifadəsinə aid kiçik bir nümunə göstərək:

```
#include <stdio.h>  
  
int main(){  
  
do{  
  
printf("Ok");  
  
} while(1==2);  
  
return 0;  
  
}
```

Gördüyümüz kimi, ilk olaraq do yazılır, fiqurlu mötərizələr açılır və bu mötərizələrin içində əməliyyatlar yazılır. Əməliyyatlar bitdikdən sonra fiqurlu mötərizə bağlanır və while(şərt) yazılır. Yuxarıdakı nümunədə şərtimiz 1-in 2-ə bərabər olmasıdır. Təbii ki, 1 və 2 bərabər deyillər, şərt ödənmir. Şərt ödənmədiyinə görə də operator öz işini bitirir. Ancaq yenə də, şərtə baxmazdan öncə operator əməliyyatları bir dəfə yerinə yetirmiş olur. Belə deyə bilərik ki,

şərtədən asılı olmadan operator əməliyyatları bir dəfə yerinə yetirir, əməliyyatları bir dəfə yerinə yetirdikdən sonra şərtə baxır.

İndi isə, daha real bir nümunə yazaq. İstifadəçiyə ədədin kvadratını tapa biləcəyi bir proqram təqdim edək. İstifadəçi bir dəfə ədədin kvadratını tapdıqdan sonra ondan yenidən hesablamaq istəyib-istəmədiyini soruşaq. Əgər istəyirsə 1, istəmirsə 2 daxil etməsini tələb edək. Sonra isə şərtə daxil olunan ədədin 1 olub-olmadığını yoxlayaq. Bir olarsa şərt ödənəcək və operator yenidən əvvələ qayıdacaq və əməliyyatları yerinə yetirəcək. Koda baxaq:

```
#include <stdio.h>

int main(){

int i,a;

do{

printf("Ədədi daxil edin:\n");

scanf("%d",&a);

printf("%d\n",a*a);

printf("Yenidən hesablama aparmaq istəyirsiniz? İstəyirsinizsə 1 daxil edin və enter basın. Yenidən hesablama aparmaq istəmirsinizsə 2 daxil edin və enter basın.\n");

scanf("%d",&i);

} while(i==1);

return 0;

}
```

Burada i və a dəyişənlərini elan edirik. Əməliyyatlara başladıqda ilk olaraq a-nı istifadəçidən qəbul edirik və kvadratını çap edirik. Sonra isə istifadəçidən yenidən əməliyyat aparmaq istəyib-istəmədiyini soruşuruq. İstifadəçi bir ədəd daxil edir. Sonda while ilə daxil olunan i dəyişəninin 1 olub-olmadığını yoxlayırıq. Ədəd 1 olarsa operator əvvələ qayıdır və əməliyyatları yerinə yetirir. 1 olmadıqda isə operator öz işini bitirir.

Funksiyalar

İlk olaraq funksiyaların istifadəsini riyaziyyatda görə bilərik. Məsələn, x kvadratı funksiyasını aşağıdakı şəkildə yazı bilərik:

$$Y=F(x)=x*x;$$

Burada, x -ə müxtəlif qiymətlər verdikdə y də müxtəlif qiymətlər alır. Məsələn, $x=2$ olduqda $y=2*2=4$ olur, $x=3$ olduqda $y=3*3=9$ olur. Təbiətdə demək olar hər şey funksiyalarla təyin olunur. Məsələn, gedilən yolu tapmaq üçün funksiya:

$$Y=F(\text{sürət,zaman})=\text{sürət}*\text{zaman}$$

Təbiətdə də bir çox qiymətlər müxtəlif funksiyalarla təyin olunur. Funksiyada ən əsas olan onun parametrləri, funksiyanın özünün riyazi düsturu və funksiyanın nəticədə alacağı qiymətdir. Məsələn, x -in kvadratı funksiyasında y alınacaq nəticə, x parametr, $x*x$ isə funksiyanın düsturudur.

Proqramlaşdırmada da funksiyaların işləmə prinsipi eynidir. Sadəcə, funksiyanı müəyyən sintaksislə yazmaq lazımdır. Bəzi proqramlaşdırma dillərində funksiyanın elan olunma qaydası daha asandır. C dilində isə, funksiyanı elan edərkən funksiyaadan alınacaq nəticənin tipini və veriləcək parametrlərin tipini qeyd etməliyik. Məsələn, int tipli bir parametrli bir funksiya yazaq. Yazacağımız funksiya x -in kvadratını tapsın.

```
#include <stdio.h>

int kvadrat(int x){

return x*x;

}

int main(){

printf("%d\n",kvadrat(5));

printf("%d\n",kvadrat(6));

printf("%d\n",kvadrat(7));

return 0;
```

}

Kodu yazdıq, indi isə izahına keçək. Yazdığımız kod daxilindəki funksiya ədədin kvadratını tapmaq üçündür. Gördüyümüz kimi, funksiyanı main funksiyasından əvvəldə yazırıq, main funksiyasının daxilində yazmırıq. Funksiyamızın adını «kvadrat» olaraq təyin etmişik. Kodda «int kvadrat(int x)» yazmışıq. Burada ilk int funksiyaadan alınan nəticənin tipini bildirir. Məsələn funksiyanın 6 rəqəminin kvadratını tapanda 36 alınacaq, 36 ədədinin tipi də tamdır, yəni int. Burada ilk int 36-nın tipini göstərəcək, yəni nəticənin. Mötərizə daxilində isə «int x» yazmışıq. Burada x funksiyaamızın parametri x-dir. Yazdığımız «int» isə x-in tipidir. Yəni x ancaq tam qiymətlər ala bilər. Bunları izah etdikdən sonra funksiyanın yerinə yetirdiyi əməliyyatlara keçək. Funksiyanın əməliyyatları «{« və «}» simvolları arasında yazılır. Funksiyanın əməliyyatları isə sadəcə «return x*x» sətrindən ibarətdir. Burada return sözü funksiyanın sonda bizə verəcəyi qiyməti bildirir. Yəni, funksiya sonda bizə «x*x» qiymətini verəcək. Ona görə də, funksiya bizə yazdığımız ədədin kvadratını verəcək.

Funksiyanın özünü yazdıqdan sonra onu main funksiyasının daxilində istifadə edə bilərik. Funksiyanı istifadə etmək üçün sadəcə adını yazırıq və funksiyaaya lazım olan parametrləri ötürürük. Yuxarıdakı nümunədə funksiyanın istiadəsi «kvadrat(5)» bu şəkildə yazılmışdır. Burada mötərizə daxilində yazdığımız 5 funksiyaaya ötürülən parametrdir. Nəticədə funksiya 5-in kvadratını, yəni 25-i bizə qaytaracaq. Yazdığımız kodda «kvadrat(5)» hissəni printf() funksiyası daxilində yazaraq çap edə bilərik. Çap etmək üçün istifadə etdiyimiz printf() funksiyasında ilk parametrin daxilində «%d» yazmağımızın səbəbi, funksiyaadan alınan qiymətin int tipində olmağıdır. Yazdığımız proqramda nəticə olaraq 5-in, 6-nın və 7-nin kvadratı çap olunacaq. Funksiyanın yazılması müəyyən mənada işimizi rahatlaşdırır. Məsələn, üç dəfə x*x yazmaq yerinə kvadrat(x) yazırıq. Burada düsturumuz sadə olduğu üçün fərq çox bilinməsə də, mürəkkəb funksiyalarda fərq çox yaxşı görünür. Müxtəlif nümunələr yazdıqca bunu daha yaxşı görəcəyik və funksiya anlayışını daha yaxşı başa düşəcəyik.

İndi isə faktorialı hesablamaq üçün bir funksiya yazaq və onu bir neçə dəfə istifadə edək:

```
#include <stdio.h>
```

```
int faktorial(int x){
```

```
int i;
```

```

int faktorial=1;

for(i=1;i<=x;i++){

faktorial=faktorial*i;

}

return faktorial;

}

int main(){

printf("%d\n",faktorial(3));

printf("%d\n",faktorial(4));

printf("%d\n",faktorial(5));

return 0;

}

```

Burada funksiya parametrlərlə faktorial tapılacaq ədəd ötürülür. Funksiyanın daxilində bir faktorial dəyişəni elan olunur və ilk qiyməti 1 olur. Həmin dəyişən 1-dən x-ə qədər bütün ədədlərə vurulur və x-in faktorialı tapılmış olur. Sonda isə return vasitəsilə funksiyanın qiyməti təyin olunmuş olur. Aşağıda isə 3, 4 və 5-in faktorialı hesablanır. Burada funksiya istifadə etmək işimizi çox rahatlaşdırır. Funksiyadan istifadə etməsəydik, 3, 4, və 5 ədədlərinin hər biri üçün faktorialı ayrıca hesablamaq olacaqdıq və işimiz uzun olacaqdı.

Fayllarla iş

C dilində və bir çox dillərdə kompyuterdəki digər fayllarla işləmək mümkündür. Bunun üçün bir sıra fayl funksiyaları mövcuddur. Bu funksiyalardan istifadə edərək yeni fayllar yarada, mövcud faylları silə və mövcud fayllar üzərində dəyişikliklər edə bilərik.

Əvvəlcə ən sadə funksiyalardan biri olan `remove()` funksiyasına baxaq. Bu funksiya vasitəsilə digər faylları silə bilərik. Məsələn, ekranımızda `sekil.jpg` adında bir fayl var və biz onu silmək istəyirik. Bunun üçün `remove` funksiyasından istifadə edə bilərik.

```
#include <stdio.h>

int main(){

remove("sekil.jpg");

printf("Fayl silindi.");

return 0;

}
```

Nəticədə `"sekil.jpg"` adındakı fayl silinir. Diqqət etmək lazımdır ki, silmək istədiyimiz faylla işlətdiyimiz proqram eyni qovluqda olsun.

Yeni fayl yaratmaq üçün isə `fopen` funksiyasından istifadə etməliyik. Nümunəyə baxaq:

```
#include<stdio.h>

int main(){

fopen("file2.txt","w");

return 0;

}
```

Burada `fopen()` funksiyası ilə `"yenifayl.txt"` adlı fayl yaradırıq. Burada funksiyanın birinci parametrinə yeni faylın adını yazmış oluruq. İkinci parametrlə olaraq isə `"w"` yazırıq. İkinci parametrlə `"w"` yazdığımız üçün funksiya yeni faylı

yaradacaq, əgər "yenifayl.txt" adlı fayl varsa faylın içini təmizləyəcək. Burada ikinci parametr müxtəlif qiymətlər ala bilər.

r və ya rb - faylı oxumaq üçün açmaq;

w və ya wb - Əgər fayl yoxdursa yeni fayl yaradır, fayl varsa içini təmizləyərək yeni bir fayl yaradır.

A və ya ab — Əgər fayl yoxdursa yaradır, varsa növbəti yazılacaq məlumatları mövcud faylın sonuna əlavə edir.

r+ və ya rb+ - Mövcud faylı həm oxumaq, həm də yazmaq üçün açır.

w+ və ya wb+ — Faylı həm oxumaq, həm də yazmaq üçün açır. Əgər fayl yoxdursa faylı yenidən yaradır, varsa içindəkiləri silir.

A+ və ya ab+ - Əgər fayl yoxdursa yaradır, varsa növbəti yazılacaq məlumatları mövcud faylın sonuna əlavə edir.

İndi isə bir fayla hər-hansı bir məlumatı yazdıraq. Bunun üçün `fprintf()` funksiyasından istifadə edəcəyik.

```
#include<stdio.h>
```

```
int main(){
```

```
FILE *fp;
```

```
fp=fopen("file.txt","w");
```

```
fprintf(fp,"Ok");
```

```
fclose(fp);
```

```
return 0;
```

```
}
```

Burada ilk öncə `FILE *fp` yazaraq `FILE` tipindən bir `fp` dəyişəni elan edirik. Dəyişənin adının əvvəlində ulduz simvolu olmasının səbəbi, dəyişənin ünvan dəyişəni olmasıdır. Ünvan dəyişənlərini növbəti mövzularda izah edəcəyik. Növbəti olaraq `fopen()` funksiyası ilə "file.txt" faylını açırıq. «`fp=fopen("file.txt","w")`» yazırıq, çünki, yazma funksiyasında bundan istifadə edəcəyik. Sonra isə `fprintf()` funksiyası ilə fayla "Ok" yazırıq. Funksiyanın birinci

parametri hansı faylı hansı formada açacağımızı göstərən «fp» dəyişənidir. Sonda isə fclose() funksiyası ilə faylı bağlayırıq.

Yuxarıdakı nümunədə proqram açılmış fayldakı məlumatları silir və yerinə yazır. Əgər, fayldakı məlumatı silməməyini, ardına yazmağını istəyiriksə, o zaman fopen() funksiyasının ikinci parametri olaraq «w» yox «a» yazmalıyıq.

```
#include<stdio.h>

int main(){

FILE *fp;

fp=fopen("file.txt","a");

fprintf(fp,"Ok");

fclose(fp);

return 0;

}
```

Bu proqramı bir neçə dəfə işə saldıqda, faylda bir neçə dəfə ard-arda "Ok" yazıldığını görəcəyik. Çünki, fopen funksiyasında ikinci parametr olaraq a yazdığımıza görə, funksiya fayldakı köhnə məlumatları silmədən, yeni məlumatları köhnə məlumatların ardına yazır.

Yazılacaq yeni məlumatın növbəti sətirdə olmasını istəyiriksə, sadəcə yazılacaq ifadənin qarşısına «\n» ifadəsini yazmağımız kifayətdir.

```
#include<stdio.h>

int main(){

FILE *fp;

fp=fopen("file.txt","a");

fprintf(fp,"\nOk");

fclose(fp);

return 0;

}
```

Nəticədə yazdığımız yeni yazı, faylda əvvəlki məlumatlardan sonra yeni sətirdə yazılacaq.

Yeni fayl yaratmaq, silmək və dəyişiklik etməyi öyrəndikdən sonra faylı oxumaq qaydasına baxaq. Faylı oxumaq üçün `fscanf()` funksiyasından istifadə edəcəyik.

```
#include<stdio.h>

int main(){

FILE *fp;

char str[100];

fp=fopen("file.txt","a+");

fscanf(fp, "%s" ,str);

fclose(fp);

printf("%s",str);

return 0;

}
```

Burada ilk olaraq bir xarakter çoxluğu yaradıırıq, sonra isə `fscanf()` funksiyası vasitəsilə məlumatları oxuyub bu dəyişənə ötürürük. Sonda isə xarakter çoxluğunu çap edirik. Burada fayl çoxluğunu çap etmək üçün `printf()` funksiyasında `"%s"` ifadəsindən istifadə edirik.

İndi isə, bu funksiyalardan istifadə edərək kiçik bir qeydiyyat proqramı yaradaq.

```
#include<stdio.h>

int main(){

FILE *fp;

char ad[10], soyad[10], yas[4];

printf("Adinizi daxil edin:");

scanf("%s",ad);

printf("Soydinizizi daxil edin:");

scanf("%s",soyad);

printf("Yasinizi daxil edin:");
```

```

scanf("%s",yas);
fp=fopen("file.txt","a");
fprintf(fp,"\n%s %s %s",ad,soyad,yas);
fclose(fp);
printf("Qeydiyyatınız tamamlandı. Tesekkurler.");
return 0;
}

```

Burada ad, soyad, və yaşı istifadəçidən qəbul edirik və fayla yazırıq. Nəticədə qeydiyyatdan keçənlərin adları bizim fayla yazılmış olur. Fprintf() daxilində dəyişənləri yazarkən, printf() funksiyasındakı kimi əvəzləyici «%s», «%d» kimi ifadələrlə yazırıq.

Kitabxana yaratmaq

Mürəkkəb bir problemi həll edərkən, problemi hissələrə bölərək onu daha rahatlıqla həll edə bilərik. Böyük bir proqram yazdığımızı fərz edək. Burada onlarla funksiyalar və digər ifadələr olacaq. Bunların hamısını bir səhifədə yazmaq aşırı qarışıqlığa yol açə bilər. Eyni zamanda, bir proqramın üzərində bir neçə nəfər işləyirsə, onların hamısının bir səhifədə işləməsi mümkünəz olacaq. Burada artıq hər proqramçı ayrı bir hissəni hazırlamalı və sonra da bu hissələr birləşdirilməlidir. Ən sadə bir nümunəyə baxaq.

```

#include <stdio.h>

int topla(int x,int y){

return x+y;

}

int main(){

printf("%d",topla(5,4));

return 0;

```

}

Burada bir ədəd toplama funksiyası yazmışıq. Ancaq burada yüzlərlə funksiya ola bilərdi. Onların hamısını isə bir səhifədə yazmaq proqramda aşırıq qarışıqlıq yaradardı. Ona görə də, funksiyanızı ayrı bir səhifədə yazıb oradan çağırmağa ehtiyac duya bilərdik.

Funksiyamınızı ayrı bir səhifə yazmaq üçün yeni bir səhifə açaq və sadəcə funksiyanı o səhifəyə yazaq.

```
int topla(int x,int y){  
  
return x+y;  
  
}
```

Yeni faylımızda sadəcə funksiyanızın kodları olacaq. Bu səhifəni «topla.h» adı ilə yadda saxlayaq. Öz yaradacağımız kitabxanalar «.h» sonluğu ilə bitir.

Əsas səhifəmizdə isə aşağıdakı kodları yazaq və hər-hansı bir adla yadda saxlayaq.

```
#include <stdio.h>  
  
#include "topla.h"  
  
int main(){  
  
printf("%d",topla(5,4));  
  
return 0;  
  
}
```

Gördüyümüz kimi, burada #include "topla.h" yazaraq öz yaratdığımız kitabxanani əsas proqramımıza çağırırıq və proqramımızda "topla.h" səhifəsinin içində olan funksiyanı işlətməmiş oluruq. Bir şeyi qeyd edək ki, "topla.h" səhifəsi ilə əsas səhifəmizi yadda saxladığımız səhifə eyni qovluqda olmalıdır.

Əgər yeni yaratdığımız kitabxana əsas proqramımızın yerləşdiyi qovluqda yox, başqa qovluqdadırsa, onda kitabxananın ünvanını yazmalıyıq. Məsələn, proqramımız masaüstündədir. Kitabxanamız isə, masatüstündəki «kitabxanalar» qovluğundadır. O zaman, kitabxanani əsas səhifəyə çağırarkən başlığı aşağıdakı şəkildə yazmalıyıq:

```
#include <stdio.h>

#include "kitabxanalar/topla.h"

int main(){

printf("%d",topla(5,4));

return 0;

}
```

İndi isə nisbətən daha mürəkkəb bir kitabxana yaradaq. Kitabxanamız fayllar işləmək üçün olsun. Kitabxanamız aşağıdakı şəkildə olacaq:

```
int create_file(char file_name[20], char file_content[100]){

FILE *fp;

fp=fopen(file_name,"w");

fprintf(fp,"%s",file_content);

fclose(fp);

return 0;

}

int delete_file(char file_name[20]){

remove(file_name);

}

int edit_file(char file_name[20], char new_information[100]){

FILE *fp;

fp=fopen(file_name,"a+");

fprintf(fp,"%s",new_information);

fclose(fp);

return 0;

}
```

Kitabxananın daxilində yeni fayl yaratmaq üçün `create_file` funksiyası, fayla yeni məlumat əlavə etmək üçün `edit_file` funksiyası faylı silmək üçün isə `delete_file` funksiyası mövcuddur. Kitabxananı "`file_library.h`" adı ilə yadda saxlayaq. İndi isə əsas proqramımıza keçək:

```
#include<stdio.h>

#include "file_library.h"

int main(){

create_file("my_file.txt","Fayla bu məlumat yazılsın");

edit_file("my_file.txt","\nyeni məlumat");

delete_file("remove_file.txt");

return 0;

}
```

Bu proqramı isə kitabxananın olduğu qovluqda yadda saxlayaq və işə salaq. Gördüyümüz kimi, kitabxana daxilində yazdığımız funksiyalardan istifadə edərək, əsas səhifədə asanlıqla əməliyyatlar apardıq. Əgər kitabxanamızı başqa bir proqramçı yazsaydı belə, bizə sadəcə funksiyanın adını və hansı parametrləri aldığını deməyi kifayət idi. Verilən məlumata əsasən biz çox asanlıqla əməliyyatlar apara bilərdik.

Struct tiplər

Gta oyununu demək olar ki, hər kəs oynayıb. Oyunda çoxlu xarakterlər var, həmçinin maşınlar. Maşınların adı, rəngi və s. kimi bir sıra xüsusiyyətləri mövcuddur. Fərz edək ki, olduğumuz bölgədə 100 ədəd maşın var. Hər maşının isə dediyimiz kimi müxtəlif xüsusiyyətləri mövcuddur. Bu xüsusiyyətləri isə qeyd etmək üçün dəyişənlərə ehtiyacımız var. Ən sadə halda hər maşın üçün 2-3 dəyişənə ehtiyacımız var. Ümumilikdə isə bizə 300 dəyişən lazım olacaq. Məsələn, «`masin1_adi`», «`masin1_rengi`», «`masin1_ili`», «`masin2_adi`», «`masin2_rengi`», «`masin2_ili`» və s. Bu şəkildə yazmaq proqramda çox qarışıqlıq yaradacaq, çünki, hər maşın üçün 3 ədəd eyni növ dəyişəni elan etməyə ehtiyac duyacağıq. Ancaq, belə bir imkanımız olsa idi, 3 növ xüsusiyyəti təmsil edən dəyişəni əvvəlcədən bir dəfə qeyd etsəydik və hər maşın üçün isə yalnız 1 dəyişən elan edib xüsusiyyətləri

yaza bilsəydik onda həm qarışıqlıq azalar, həm də proqramı yazmağımız daha rahat olardı. C dilində bunu structlar vasitəsilə edə bilirik. Sadə bir nümunə göstərək.

```
#include <stdio.h>

struct cars{

char name[20];

char color[20];

int year;

};

int main(){

struct cars car1={"Bmw","red",2003};

struct cars car2={"Mercedes","red",2003};

struct cars car3={"Toyota","red",2003};

struct cars car4={"Audi","red",2003};

struct cars car5={"Honda","red",2003};

printf("Üçüncü maşının rəngi: %s",car3.color);

return 0;

}
```

Yazdığımız kodu izah etməyə çalışaq. İlk öncə qeyd edək ki, struct tipini main funksiyasından əvvəl yazırıq. Əvvəlcə «struct cars{» yazaraq structurumuzu yaratmış oluruq. Structumuzun içində isə üç dəyişən elan edirik, adı, rəngi və ili. Bunları elan etdikdən sonra structumuzu bağlayırıq. Structu bağladıqdan sonra «;» simvolu qoyulduğunu unutmayaq. Structu yaratdıqdan sonra main funksiyasında bu struct tipindən obyektlər yaratmalıyıq. «Struct cars car1» yazaraq cars tipindən «car1» obyektini yaratmış oluruq. Sonra isə «=» işarəsi qoyaraq «{» və «}» fiqurları arasında sıra ilə obyektin dəyişənlərinin qiymətlərini yazırıq. İlk qiymət maşının adı, növbəti qiymət rəngi, növbəti qiymət isə maşının ilidir. Burada bu tipdən 5 ədəd maşın elan edirik. Sonra isə printf ilə obyektin dəyişənini çap edirik. Məsələn, «car3» dəyişəninin rəngini götürmək üçün «car3.color» yazmalıyıq.

Burada bu dəyişən xarakter çoxluğu olduğu üçün printf() funksiyasında «%s» ifadəsindən istifadə edərək xüsusiyyəti çap etmiş oluruq.

İndi isə fərz edək ki, proqram daxilində çoxlu sayda nəfərin xüsusiyyətlərini qeyd edəcəyik. Bunu bir nəfər üçün aşağıdakı şəkildə edə bilərik.

```
#include <stdio.h>

struct human{

char ad[20];

int yas;

int boy;

int ceki;

};

int main(){

struct human human1={"Shukur",23,178,75};

printf("Human1: %s",human1.ad);

return 0;

}
```

İlk öncə human adında bir struct yaradıırıq, sonra içinə lazım olan dörd dəyişəni qeyd edirik, sonra isə main hissəsində yaratdığımız struct tipindən obyekt yaradıb onun xüsusiyyətlərini təyin edirik. Sonda isə xüsusiyyətlərdən birini çap edirik.

Structlarla bağlı kiçik bir nümunə daha yazaq. Bu dəfə telefonlarımız olsun və onların xüsusiyyətlərini bir struct vasitəsilə yazaq.

```
#include <stdio.h>

struct telefon{

char ad[20];

int qiymeti;

};
```

```

int main(){

struct telefon telefon1={"Smasung",300};

struct telefon telefon2={"Iphone",700};

printf("Telefon 1-in qadi ve qiymeti: %s %d azn",telefon1.ad,telefon1.qiymeti);

return 0;

}

```

Gördüyümüz kimi, telefonlar üçün bir struct yaradıırıq, həmin struct tipindən iki ədəd telefon yaradıırıq və həmin telefonların xüsusiyyətlərini qeyd edirik.

Praktika üçün bir neçə müxtəlif struct yaratmağınız məsləhət görülür. Məsələn, heyvanlara, müxtəlif qurğulara, komputerlərə aid structlar yaradın, onların xüsusiyyətlərini təyin edin. Xüsusiyyətləri istifadəçidən qəbul etmək, fayla yazmaq kimi müxtəlif əməliyyatlar edərək mövzunu daha yaxşı mənimsəməyə çalışın.

Typedef ifadəsi

Bu ifadə mövcud tiplərə qısa, alternativ adlar vermək üçün istifadə olunur. Məsələn:

```

#include <stdio.h>

int main(){

typedef int tam;

tam a=5;

printf("%d",a);

return 0;

}

```

Bu proqramda «**typedef int tam;**» yazaraq «int» sözünü tam ilə əvəz etmiş oluruq. Artıq «int» tipli dəyişən elan edərkən int əvəzinə tam sözünü də yaza

bilərik. Bu nümunədə çox fərq gözə çarpmasa da, nisbətən daha böyük ifadələrdə typedef ifadəsinin işimizi rahatlaşdırdığını görə bilərik. Məsələn, typedef ifadəsini struct tiplər üçün istifadə edək.

```
#include <stdio.h>
```

```
typedef struct telefon{
```

```
char ad[20];
```

```
int qiymeti;
```

```
} phone;
```

```
int main(){
```

```
phone telefon1={"Smasung",300};
```

```
printf("Telefon 1-in qadi ve qiymeti: %s %d azn",telefon1.ad,telefon1.qiymeti);
```

```
return 0;
```

```
}
```

Burada struct sözünün əvvəlinə «typedef» yazırıq, structdan sonra isə «phone» açar sözünü yazırıq. Artıq, main hissəsində bu struct tipindən obyekt elan edərkən «struct telefon telefon1» yazmaq yerinə qısa şəkildə «phone telefon1» yazmağımız kifayət edir.

Yeni dəyişən tipi

Biz yazdığımız nümunələrdə int, float, char kimi dəyişən tiplərindən istifadə etdik. Məsələn, int tipi tam qiymətləri alırdı, float onluq qiymətləri, char simvolları. Ancaq, bunlardan əlavə biz özümüz də yeni dəyişən tipləri yarada bilərik. Məsələn, ancaq rəng adlarını alan bir dəyişən tipi yaratmağımız mümkündür. Nümunəyə baxaq.

```
#include <stdio.h>
```

```
enum color{red,green,blue};
```

```

int main(){

enum color masinin_rengi=red;

if(masinin_rengi==red){

printf("Masinin rengi qirmizidir.");

}

return 0;

}

```

Yazdığımız kodun izahına keçək. İlk öncə enum açar sözü ilə color adında yeni dəyişən tipimizi elan edirik. «{« və »}» simvolları daxilində isə bu dəyişən tipimizdən olan dəyişənlərin hansı qiymətləri alacağını yazırıq. Bu tipdən olan dəyişənlərimiz yalnız red green və blue qiymətlərini ala bilər. Dəyişən tipimizi yaratdıqdan sonra main funksiyasının daxilində enum color masinin_rengi=red; yazaraq bu tipdən masinin_rengii adında bir dəyişən elan edirik və dəyişənə red qiymətini veririk. Sonra isə if vasitəsilə dəyişənin bu qiyməti alıb-almadığını yoxlayırıq.

Yazdığımız nümunədə yeni dəyişən elan etmək o qədər də əhəmiyyətli görünməsə də, nisbətən böyük proqramlarda bunun əhəmiyyətini görmək mümkün olar. Ən sadə halda oyundakı maşının sağa, sola, yuxarı yoxsa aşağı getməsini yadda saxlamaq üçün istiqamət adlı yeni bir dəyişən tipi yaradıb sağ, sol, yuxarı və aşağı qiymətlərini verə bilərik. Məsələn:

```

#include <stdio.h>

enum direction{left,right,up,down};

int main(){

enum direction istiqamet=left;

if(istiqamet==left){

printf("Masin sola gedir.");

}

return 0;

```

```
}
```

Böyük proqramlarda yeni yaradacağımız tiplərlə kodlaşdırmanı rahatlaşdırma və yazdığımızı kodu daha oxunaqlı edə bilərik.

Əvvəlki mövzuda qeyd etdiyimiz typedef ifadəsini buraya tətbiq edərək yeni dəyişənin elanını daha asanlaşdırma bilərik. Məsələn:

```
#include <stdio.h>

typedef enum direction{left,right,up,down} direct;

int main(){

direct istiqamet=left;

if(istiqamet==left){

printf("Masin sola gedir.");

}

return 0;

}
```

Gördüyümüz kimi, «enum» ifadəsindən əvvəl «typedef» yazırıq sonda isə «direct» adını yazırıq. Əsas hissədə isə «direct istiqamet» yazaraq dəyişəni elan edirik.

Makroslar

C dilində makros adlandırılan bir sıra elementlər mövcuddur ki, onlar vasitəsilə müəyyən əməliyyatları edə bilərik.

#define makrosu.

Bu makros proqramın gedişi ərzində bir ifadəni başqası ilə əvəzləmək üçün nəzərdə tutulur. Məsələn:

```
#include <stdio.h>

#define x 5
```

```
int main(){
printf("%d",x);
return 0;
}
```

Burada #define x 5 yazırıq və proqram öz daxilində yazılan x simvollarını 5 edir. Nəticədə 5 çap olunur.

#undef makrosu.

Bu makros, define makrosu ilə təyin olunmuş qiyməti ləğv etmək üçün istifadə olunur. Aşağıdakı kod baxaq:

```
#include <stdio.h>

#define x 5

int main(){
printf("%d",x);

#undef x

#define x 6

printf("%d",x);

return 0;
}
```

Burada x-in qiyməti bir dəfə çap olunduqdan sonra undef vasitəsilə ləğv olunur və define vasitəsilə x-ə yeni qiymət verilir.

C dilində bir sıra təyin olunmuş ifadələr mövcuddur. Bunlara çap edərək baxaq.

```
#include <stdio.h>

int main() {
printf("File :%s\n", __FILE__ );
printf("Date :%s\n", __DATE__ );
}
```

```
printf("Time :%s\n", __TIME__ );
printf("Line :%d\n", __LINE__ );
printf("ANSI :%d\n", __STDC__ );
}
```

Burada `__FILE__` əvəzetməsi faylın adını, `__DATE__` ayın tarixini, `__TIME__` zamanı, `__LINE__` proqramdakı sətir sayını göstərir. `__STDC__` isə, kompilyator ANSI standartlarına uyğun olduqda 1 qiymətini qaytarır.

Yaddaş idarəetməsi və ünvan dəyişənləri

İstifadə etdiyimiz müxtəlif proqramların müxtəlif ram tələb etdiyini görürük. Məsələn, Gta 4 gb ram tələb edir, Photoshop 2gb-dan yuxarı tələb edir və s. Bəs nəyə görə müxtəlif proqramlar müxtəlif ramlar tələb edir? Çünki, hər proqramın kodu, istifadə etdiyi resursları fərqlidir. Proqramın istifadə etdiyi resurslar həmin proqramın istifadə etdiyi rama birbaşa təsir göstərir.

Proqram/oyun proqramlaşdırılarkən müəyyən vasitələrlə rama qənaət edilməsi mümkündür. Bununla da proqramımız daha yaxşı ola bilər. Məsələn, ramı 2gb-dan yuxarı tələb ediriksə, ramı 1gb olan istifadəçi bizim proqramımızdan istifadə edə bilməyəcək. Ancaq, müəyyən üsullarla ramın istifadəsini azalda bilsək, onda 1 gb ramı olan istifadəçilər də, bizim proqramımızdan istifadə edə biləcək. Ona görə də, bu mövzu ən vacib mövzulardandır. Ancaq, bu mövzunun digər mövzulara nisbətən daha çətin olduğunu deyə bilərik.

İlk öncə dəyişənlərə diqqət edək. Məsələn, char tipli bir dəyişənin tutduğu yer 1 baytdır. Biz 1000 ədəd char tipli dəyişən elan etdik, istifadə etdik, ancaq daha sonra proqramın gedişində bu dəyişənlər bizə lazım deyil. Ancaq, onları əvvəlcədən elan etdiyimizə görə artıq onlar ramda saxlanır və 1000 bayt, təxminən 1 meqabayt yer tutur. Kiçik proqramlarda 1 meqabayt elə də böyük görünməsə də, böyük layihələrdə bu çox böyük fərqlər yarada bilər. Bu dəyişənlər artıq bizə lazım deyil, ancaq, ramda 1 meqabayt yer tutur. Ancaq, biz bu dəyişənləri ramdan silə bilsək necə? Onda ramın istifadəsi 1 meqabayt azalacaq. Ancaq, əvvəl öyrəndiyimiz adi dəyişənlərlə bunu edə bilmirik. Bunun üçün də, əvvəlcə bunu edə biləcəyimiz **ünvan dəyişənləri** ilə tanış olmalıyıq.

Qeyd edək ki, dəyişənlər proqram işlədiyi müddətdə ramda yer tutur. Proqram bağlandıqdan sonra dəyişənlər avtomatik olaraq ramdan silinir. Biz proqram işlədiyi müddətdə rama qənaət etməyə ehtiyac duyuruq.

Ünvan dəyişənləri də, adi dəyişənlər kimi elan olunur, sadəcə, dəyişənin adının əvvəlinə * simvolu qoyulur. Məsələn:

```
int *x;
```

Ünvan dəyişənlərini bu şəkildə elan edirik. Ünvan dəyişənini istifadə etdiyimiz kiçik bir nümunəyə baxaq:

```
#include <stdio.h>

int main(){

int *x;

*x=5;

printf("Dəyişən: %d\n",*x);

printf("Dəyişən: %p\n",&x);

return 0;

}
```

Gördüyümüz kimi, adi dəyişən kimi ünvan dəyişənini də elan edib printf funksiyası ilə çap edə bilirik. İkinci çap etdiyimiz isə, dəyişənin yaddaşda saxlandığı ünvandır. Ünvan printf funksiyasında çap olunarkən «%p» əvəzləyicisindən istifadə olunur. Adından da göründüyü kimi, ünvan dəyişəni özündə ünvanı saxlayır.

Ünvan dəyişənini char tipindən xarakter çoxluğu kimi də istifadə etmək mümkündür. Məsələn:

```
#include <stdio.h>

#include <string.h>

int main(){

char *x;
```

```

x="Salam";

printf("%s",x);

return 0;

}

```

Gördüyümüz kimi, əlavə çoxluq yaratmadan char tipli ünvan dəyişəninə bütöv bir sözü yazı bildik. Artıq, sözləri xarakter çoxluğu yaratmadan ünvan dəyişəninə ötürə bilərik.

Yaratdığımız ünvan dəyişəni ilə yaddaşdan müəyyən qədər yer ayıra bilərik. Həmin ayırdığımız yerdəki məlumatları işimiz bitdikdən sonra onu silə bilərik. Bununla rəqəmə qənaət etmiş oluruq. Bunun üçün **malloc** və **free** funksiyalarından istifadə edəcəyik. Bu funksiyalar stdlib.h kitabxanasına daxildir. Əvvəlcə **malloc** funksiyasının istifadə qaydasına baxaq.

```

#include <stdio.h>

#include <stdlib.h>

int main(){

int *x;

x=malloc(5);

*x=5;

printf("%d\n\n",*x);

return 0;

}

```

Yuxarıda qeyd etdiyimiz kimi, malloc funksiyası stdlib.h kitabxanasına daxil olduğu üçün, proqrama stdlib.h kitabxanasını əlavə edirik. x=malloc(5) yazdıqda, x üçün yaddaşda 5 bayt yer ayrılmış olur. Növbəti sətirdə *x=5 yazaraq x-ə 5 qiymətini ötürürük. Ancaq burada biz dəyişəni silməmişik, yəni hələdə dəyişən üçün yaddaşda 5 bayt yer ayrılıb. Biz artıq həmin dəyişəndən istifadə etməyəcəyiksə, onu **free** funksiyası ilə yaddaşdan silə bilərik.

```

#include <stdio.h>

```

```

#include <stdlib.h>

int main(){

int *x;

x=malloc(5);

*x=5;

// Fərz edək ki, artıq x dəyişənindən müəyyən əməliyyatlarda istifadə olundu.

free(x);

printf("%d",*x);

// Artıq ramda ayrılmış yer silinmişdir. Növbəti əməliyyatlarda ramda daha az
məlumat olacaq.

return 0;

}

```

Burada `free(x)` yazaraq `x` üçün ayrılmış sahəni ramdan silirik. Beləliklə də, rama qənaət etmiş oluruq. Artıq `free(x)` sətrindən sonrakı əməliyyatlar yerinə yetirilən vaxt ramdan daha az istifadə olunacaq. Eyni zamanda, proqramı işlətdikdən sonra 0 çap olunduğuna diqqət edək. Bu `x`-in son qiymətidir, çünki daha əvvəl `x` üçün ayrılmış yer ramdan silinmişdir, oradakı məlumatlar da silinmişdir.

Yaddaş idarəetməsi üçün **malloc** funksiyasından əlavə **calloc** və **realloc** funksiyası da mövcuddur.

Calloc funksiyasının istifadəsi, **malloc** funksiyasının istifadəsinə oxşardır. Bu funksiya, hamısı eyni ölçüdə və sıfırlanmış `n` sayda təyin edilmiş ölçüdə yer ayırır. Funksiya iki parametrlə alır, birinci parametrlə ayrılacaq yerin sayı, ikinci parametrlə isə hər bir yerin tutduğu yaddaşdır.

```

#include <stdio.h>

#include <stdlib.h>

int main(){

int *x;

```

```

x=calloc(5,4);

*x=5;

printf("%d",*x);

return 0;

}

```

Realloc funksiyası, əvvəlcədən ayrılmış yerin ölçüsünü dəyişmək üçün istifadə olunur. Məsələn, x üçün əvvəl 5 bayt yer ayırmışdır, sonra isə bu yerin ölçüsünü 10 bayt etmək istəyiriksə, realloc funksiyasından istifadə edirik. Bu funksiya iki parametr alır, birinci parametr yer ayrılmış dəyişənin adı, ikinci isə yeni ayrılmış yerin yeni ölçüsü.

```

#include <stdio.h>

#include <stdlib.h>

int main(){

int *x,*y;

x=malloc(5);

*x=5;

y=realloc(x,10);

return 0;

}

```

String.h kitabxanası

C dilində, yazılarla, sətirlərlə işləmək üçün bir sıra funksiyalar mövcuddur ki, bu funksiyalar da string.h kitabxanasına daxildir. Bu funksiyalara məlumatlardan istifadə edərkən ehtiyac duyula bilir. Bunlardan bir neçəsinə baxaq.

1) strcpy() funksiyası — Bu funksiya, müəyyən bir yazını müəyyən bir dəyişənə kopyalamaq üçün istifadə olunur. Məsələn:

```

#include <stdio.h>

```

```
#include <string.h>

int main(){

char a[50];

strcpy(a,"ok");

printf("%s",a);

return 0;

}
```

Funksiya iki parametrl alır. Birinci parametrl yazının kopyalanacağı dəyişən, ikinci parametrl isə kopyalanacaq yazı olur. Nəticədə, «ok» yazısı a dəyişəninə köçürülmüş olur.

2) strlen() funksiyası - Yazının uzunluğunu ölçmək üçün istifadə olunur. Funksiyanın istifadə qaydası aşağıdakı şəkildədir:

```
#include <stdio.h>

#include <string.h>

int main(){

char a[50];

int uzunluq;

strcpy(a,"ok");

uzunluq=strlen(a);

printf("Yazinin uzunlugu: %d",uzunluq);

return 0;

}
```

Gördüyümüz kimi, burada əvvəlcə a dəyişəninə «ok» yazısı ötürülür, sonra isə uzunluq dəyişəninə strlen funksiyası vasitəsilə yazının uzunluğu ötürülür və çap olunur. Çap olunan qiymət 2 olur, çünki, "ok" yazısı 2 simvoldan ibarətdir.

3) memchr() funksiyası() - Göstərilən simvoldan sonrakı hissəni tapmaq üçün istifadə olunur. Məsələn:

```

#include <stdio.h>

#include <string.h>

int main () {

char yazi[100];

char *son;

strcpy(yazi,"Salam. Bu kitab C diline aiddir. C dili.");

son=memchr(yazi,'.',strlen(yazi));

printf("%s", son);

return(0);

}

```

Funksiya üç qiymət alır, birinci qiymət yazı, ikinci qiymət simvol, üçüncü qiymət isə yazının uzunluğudur. Burada simvol nöqtə olduğuna görə, funksiya ilk nöqtədən sonrakı hissəni qaytarmış olur.

3) strcmp() funksiyası — İki yazını müqayisə etmək üçün istifadə olunur. Məsələn:

```

#include <stdio.h>

#include <string.h>

int main(){

char str1[50];

char str2[50];

int result;

strcpy(str1,"b");

strcpy(str2,"a");

result=strcmp(str1,str2);

printf("%d",result);

return 0;

```

```
}
```

Burada str1 dəyişəninə b, str2 dəyişəninə isə a ötürülür. İlk dəyişənimiz str1 dəyişəninə b hərfi əlifba sırasında a-dan sonra gəlir, yəni a-dan böyük sayılır. Ona görə də str1 dəyişəni str2-dən böyük sayılır. Əgər «str1» dəyişəni str2-dən böyük sayılarsa 1 qiyməti qaydır. Əksinə olarsa isə 0-dan kiçik qiymət qaydır. Bərabər olarsa isə 0 qaydır. Yazdıqlarımızı şərtlərlə yoxlayaraq müəyyən əməliyyatlar aparmaq mümkündür.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main(){
```

```
char str1[50];
```

```
char str2[50];
```

```
int result;
```

```
strcpy(str1,"b");
```

```
strcpy(str2,"a");
```

```
result=strcmp(str1,str2);
```

```
if(result>0){
```

```
printf("Birinci boyukdur.");
```

```
}
```

```
else if(result<0){
```

```
printf("ikinci boyukdur.");
```

```
}
```

```
else{
```

```
printf("Beraberdur.");
```

```
}
```

```
return 0;
```

```
}
```

Gördüyümüz kimi, qayıdan qiymət şərtlə yoxlanaraq müəyyən əməliyyatlar aparılır.

4) `strncmp()` funksiyası — Bu funksiyada `strcmp()` funksiyasına oxşar olaraq iki yazını müqayisə edir. Ancaq, burada müqayisə olunacaq simvol sayı da qeyd olunur. Məsələn:

```
#include <stdio.h>

#include <string.h>

int main(){

char str1[50];

char str2[50];

int result1,result2;

strcpy(str1,"bayram");

strcpy(str2,"baba");

result1=strcmp(str1,str2);

result2=strncmp(str1,str2,2);

printf("Strcmp: %d \n Strncmp: %d", result1, result2);

return 0;

}
```

Burada həm `strcmp` funksiyasından, həm də `strncmp` funksiyasından istifadə etdik. Birinci işlətdiyimiz `strcmp` funksiyası nəticədə 1 qaytardı, çünki, bayram sözü əlifbada baba sözündən sonradı. İlk iki hərf eyni olsa da, birinci ifadədəki y hərfi ikinci ifadədəki b hərfindən sonra gəldiyi üçün böyük sayılır, nəticədə `str1` böyük sayılır və funksiya 1 qiymətini qaytarır.

İkinci istifadə etdiyimiz `strncmp()` funksiyasında isə, nəticə olaraq 0 qayıdır. Çünki, funksiyada üçüncü parametr olaraq 2 yazmışıq, yəni, yalnız iki xarakter müqayisə olunacaq. İfadələrdə isə ilk iki hərf eyni olduğuna görə, hər ikisində ilk iki simvol «ba» olduğuna görə, funksiya 0 qiymətini qaytarır.

5) strtok() funksiyası — Yazılmış bir ifadəni, hər-hansı bir simvoluna görə hissələrə bölməyə imkan verir. Məsələn, bir cümləni sözlərinə görə ayırmaq istəyirik. O zaman, bu funksiyadan istifadə edəcəyik. Nümunə kodu göstərək.

```
#include <stdio.h>

#include <string.h>

int main(){

char x[50];

char *y;

strcpy(x,"Bu yazi hisseler bolunecek");

y=strtok(x," ");

printf("%s",y);

return 0;

}
```

Bu nümunədə funksiya yazdığımız cümləni boşluq simvoluna görə bölərək ilk hissəni qaytarır. Nəticədə «Bu» sözü çap olunur.

Cümləni hissələrə ayırdıqdan sonra bütün hissələri çap etmək istəyiriksə, onda aşağıdakı koddan istifadə edə bilərik:

```
#include <stdio.h>

#include <string.h>

int main(){

char x[50];

char *y;

strcpy(x,"Bu yazi hisseler bolunecek");

y=strtok(x," ");

while(y!=NULL){
```

```
printf("%s",y);  
y=strtok(NULL," ");  
}  
return 0;  
}
```

Nəticədə cümlə hissələrə bölünərək çap olunur və hər hissədən sonra vergül qoyulur.

C++ dilinə keçid

C++ dili, C dilinin təkmilləşdirilmiş formasıdır. C və C++ dilləri bir-birinə çox oxşardır. Demək olar ki, bütün C kodları C++ dilində də işləməkdədir. C++ üçün əlavə bir kompilyator yükləməyə ehtiyac yoxdur. İstifadə etdiyiniz Dev C++ həmçinin C++ kodlarını da kompilyasiya edəcək.

C++ dili özü ilə bir çox yeniliyi gətirir. Bu yeniliklərin ən mühümlərindən biri də obyekt yönümlü proqramlaşdırma. Obyekt yönümlü proqramlaşdırma müasir dillərin bir çoxunda mövcuddur və hal-hazırda çox istifadə olunmaqdadır.

C++ dilindəki yeniliklərdən biri də ad fəzalarıdır. Fərz edək ki, bir proqramı iki nəfər yazır. Bir proqramçının elan etdiyi dəyişənin adı ilə digərinin elan etdiyi dəyişənin adı eyni olarsa xəta baş verəcək. Bunun üçün də, ad fəzalarından istifadə edə bilərik. Ad fəzaları, eyni adda dəyişənlər funksiyalar yararmağa imkan verir. Fəzanı yaratmaq üçün **namespace** açar sözündən istifadə olunur. Bu açar söz yazılır və sonra isə fəzanın adı yazılır. Sonra isə, «{» və «}» işarələri arasında dəyişənlər, funksiyalar elan olunur. Məsələn:

```
#include <stdio.h>
```

```
namespace feza1{
```

```
int a;
```

```
}
```

```
namespace feza2{
```

```
int a;
```

```
}
```

```
int main(){
```

```
feza1::a=5;
```

```
feza2::a=6;
```

```
return 0;
```

```
}
```

Gördüyümüz kimi, iki fəza elan edərək hər birində a adında dəyişən yaradırıq. Bununla da programımızda heç bir qarışıqlıq yaranmır. Bir fəzanın elementinə müraciət edərkən isə, fəzanın adını yazıb :: işarələrini qoyuruq və dəyişənin, funksiyanın adını yazırıq. Buna görə də, fəza1-in a dəyişəninə müraciət edərkən feza1::a=5; yazmışıq. Artıq burada fəza1-in a dəyişəni ilə fəza2-nin a dəyişəni ayrı-ayrı dəyişənlərdir.

C++ dilindəki bir yenilik də <iostream> kitabxanasıdır. Kitabxananın daxilində giriş-çıxış operatorları, string tipi və s. mövcuddur. Kitabxananın daxilində, printf və scanf funksiyalarını əvəzləyən cout və cin obyektləri mövcuddur. Bu operatorlar «std» adlı fəzaya daxildir. Ona görə də, cout və cin operatorlarından istifadə edərkən, std::cin və std::cout yazırıq. Onların istifadəsinə baxaq.

```
#include <iostream>
```

```
int main(){
```

```
int x;
```

```
std::cin>>x;
```

```
std::cout<<x;
```

```
return 0;
```

```
}
```

Gördüyümüz kimi, cout istifadə edərkən, dəyişəni çap etmək üçün, onu mötərizə daxilində yox, « işarələrini qoyduqdan sonra yazırıq. Cin istifadə edərkən isə, » simvollarından istifadə edirik. Bu işarələri növbəti mövzularımızda izah edəcəyik.

Std fəzasının bir elementi də, **string** dəyişən tipidir. C dilində mətnləri, ifadələri saxlamaq üçün char tipli çoxluqlardan istifadə edirdik. C++ dilində isə, ifadələri saxlamaq üçün ayrıca string tipi mövcuddur. Bu tiptən rahatlıqla istifadə edə bilərik. Bu tipi aşağıdakı şəkildə istifadə edə bilərik.

```
#include <iostream>
```

```
int main(){
```

```
std::string a;
```

```

a="Salam";

std::cout<<a;

return 0;

}

```

Proqram daxilində **std** fəzasına aid elementləri yazarkən, hər dəfə «std::» yazmamaq üçün, proqramın əvvəlində **using namespace std;** yazı bilərik.

```

#include <iostream>

using namespace std;

int main(){

string a;

a="Salam";

cout<<a;

return 0;

}

```

Gördüyümüz kimi, artıq hər dəfə «std::» yamağımıza ehtiyac qalmır.

C++ dilindəki ən əsas yeniliklərdən biri də obyekt yönümlü proqramlaşdırma. Obyekt yönümlü proqramlaşdırma komanda işini rahatlaşdırır və kodu daha da oxunaqlı edir.

Öyrənəcəyimiz əsas anlayış sinif anlayışıdır. Strukt tipləri xatırlayaq. Bir elementə aid dəyişənlərimizi struktun içində saxlayırdıq. Məsələn, 5 ədəd evimiz var, bu evlərin hər birinin eni və uzunluğu var. 5 ədəd evin eni və uzunluğu üçün 10 ədəd dəyişən elan etmək yerinə, bir structun içində en və uzunluq dəyişənləri elan edirdik. Sonra isə bu struct tipindən 5 ədəd ev yaradaraq evin uzunluğunu və enini təyin edirdik.

```

#include <stdio.h>

struct ev{

int en;

```

```

int uzunluq;

};

int main(){

struct ev ev1,ev2,ev3,ev4,ev5;

ev1.en=5;

ev1.uzunluq=10;

// Digər evlər üçün də yazdığımızı fərz edək.

return 0;

}

```

Elan etdiyimiz evlər üçün ortaq olan bir şey də, hər birinin sahəsinin tapılması üçün eninin uzunluğuna vurulmasıdır. Bir neçə ev üçün eyni əməliyyatı yerinə yetirəcəyimiz üçün, bunu funksiya şəklində yazmaq ən uyğundur. Təyin etdiyimiz evlərə uyğun olaraq, dəyişənlərlə bərabər funksiyaların da bir yerdə saxlanması çox yaxşı olar. Burada artıq, həm dəyişənləri, həm də funksiyaları bir yerdə saxlamaq üçün strukt tipdən yox, **siniflərdən** istifadə edəcəyik. Siniflərin özünə xas bir çox əlavələri mövcuddur. Sadə bir sinifin istifadəsinə baxaq:

```

#include <iostream>

class Ev{

public:

int en,uzunluq;

};

int main(){

Ev ev1;

ev1.en=5;

std::cout<<ev1.en;

return 0;

}

```

Siniflər, **class** açar sözü ilə elan olunur. Gördüyümüz kimi, **class Ev{};** yazaraq sinifi elan etmiş oluruq. { ve } daxilində isə sinifin dəyişənləri, funksiyaları yer alır. Sinifi elan etdikdən sonra, int tipindən en və uzunluq dəyişənlərini elan etmişik. Ancaq, ondan əvvəlki sətirdə **public:** yazdığımıza diqqət edək. Bunun mənasını irəlidə izah edəcəyik. Hələlik isə sinif məntiqini anlamağa çalışaq.

Sinifi elan etdikdən sonra main funksiyası daxilində **Ev.ev1** yazaraq **Ev** sinifindən **ev1** obyektini elan edirik. Sonra isə **ev1.en** yazaraq, Birinci evinini təyin edirik və sonda onu çap edirik.

İndi isə, daxilində evin sahəsini hesablayan funksiyanı da saxlayan bir sinif yaradaq və onu izah edək.

```
#include <iostream>

class Ev{

public:

int en,uzunluq;

int sahe(){

return this->en*this->uzunluq;

}

};

int main(){

Ev ev1,ev2;

ev1.en=5;

ev1.uzunluq=10;

std::cout<<ev1.sahe();

std::cout<<"\n\n\n";

ev2.en=6;

ev2.uzunluq=7;

std::cout<<ev2.sahe();
```

```
return 0;

}
```

Burada dəyişənlərdən əlavə funksiya da elan etmişik. Elan etdiyimiz funksiya diqqət yetirək.

```
int sahe(){

return this->en*this->uzunluq;

}
```

Burada **this->en** eyni bir obyektin eni, **this->uzunluq** isə eyni obyektin uzunluğudur. Funksiyamız bunların hasilini bizə qaytarır. Məsələn, biz ev1 obyektini elan edərək ev1.en=5, ev1.uzunluq=10 yazdıqdan sonra, funksiyanı çağırıqda, ev1 üçün sahə funksiyaımızın qiyməti avtomatik olaraq $5 \cdot 10 = 50$ olmuş olur. Sonra isə, ev2 obyektini elan edib ev2.en=6, ev2.uzunluq=7 yazdıqda və sonra funksiyaımızı çağırıqda, ev2 üçün funksiyaımızın qiyməti avtomatik olaraq $6 \cdot 7 = 42$ olmuş olur. Funksiyanı hansı obyektə çağırırsaq, funksiya həmin obyektin dəyişənlərini istifadə edir. Burada, funksiyanı ev1 obyektinə çağırıqda, funksiya ev1-in en və uzunluq dəyişənlərindən istifadə edərək hesablama aparır. Funksiyanı ev2 obyektinə çağırıqda isə, funksiya ev2 üçün olan en və uzunluqdan istifadə edir.

Yazdığımız nümunəyə oxşar başqa bir nümunə də yazaq. Fərz edək ki, maşınlar aid dəyişənlər elan etməliyik. Dəyişənlərə maşının sürətini və getdiyi zamanı daxil etməliyik. Eyni zamanda bir funksiya ilə gedilən yolu tapmalıyıq. Maşının sürəti v , getdiyi zaman t olarsa, getdiyi məsafə $S = v \cdot t$ olacaq. Program isə aşağıdakı şəkildə olacaq:

```
#include <iostream>

class Cars{

public:

int speed,time;

int distance(){

return this->speed*this->time;

}
```



```

};

int main(){

Cars car1, car2;

car1.speed=40;

car1.time=200;

std::cout<<car1.distance();

std::cout<<"\n\n\n";

car2.speed=50;

car2.time=20;

std::cout<<car2.distance();

return 0;

}

```

Nəticədə, dəyişənlərə uyğun olaraq «car1» və «car2» obyektləri üçün gedilən yol hesablanaraq çap olunacaq.

Private etiketi. Yuxarıdakı nümunələrdə sinif daxilində yazdığımız bütün funksiya və dəyişənləri public etiketi altında yazdıq. Public etiketi altında yazdığımız funksiyaları main hissəsində istifadə edə, dəyişənlərə isə main hissəsində qiymət verə bilirdik. Ancaq, elə ola bilər ki, istəyə bilərik ki, dəyişənlərə yalnız sinif daxilində qiymət verilə bilsin, funksiyalardan isə, yalnız sinif daxilində istifadə oluna bilsin. Bu halda, funksiya və dəyişənləri public etiketi altında yox, private etiketi altında istifadə etməliyik. Aşağıda koda baxaq:

```

#include <iostream>

using namespace std;

class sinif{

public:

int x;

int funksiya(){

```

```

cout<<this->x;;

return 0;

}

};

int main(){

sinif obyekt1;

obyekt1.x=5;

obyekt1.funksiya();

return 0;

}

```

Burada sınıf və dəyişən public etiketi altında təyin olunub və main hissəsində çağırılaraq normal şəkildə işləyəcək. Aşağıdakı kod isə **işləməyəcək**.

```

#include <iostream>

using namespace std;

class sınıf{

private:

int x;

int funksiya(){

cout<<this->x;;

return 0;

}

};

int main(){

sınıf obyekt1;

obyekt1.x=5;

```

```
obyekt1.funksiya();  
  
return 0;  
  
}
```

Səbəbi isə, private etiketi altında yazılmış dəyişənə kənarda qiymət verməyə çalışılması, funksiyanın isə kənarda istifadə olunmaq istənilməsidir. Ancaq, dediyimiz kimi, private etiketi altında yazılmış funksiyalar yalnız sinifin öz daxilində istifadə oluna bilər və private etiketi altında yazılmış dəyişənlərə yalnız sinif daxilində qiymət verilə bilər. Aşağıdakı kodumuz isə normal şəkildə işləyəcək:

```
#include <iostream>  
  
using namespace std;  
  
class sinif{  
  
private:  
  
int x;  
  
int funksiya_private(){  
  
cout<<"this->x";  
  
return 0;  
  
}  
  
public:  
  
int funksiya_public(){  
  
this->x=5;  
  
funksiya_private();  
  
return 0;  
  
}  
  
};  
  
int main(){  
  
sinif obyekt1;
```

```

obyekt1.funksiya_public();

return 0;

}

```

Burada biz private etiketi altında elan etdiyimiz dəyişənə, public etiketi altındakı bir funksiyanın içində qiymət veririk və həmin funksiyanın içində, public etiketinin altında elan edilmiş funksiyanı çağırırıq. Nəticədə, **funksiya_public** adlı funksiyaımız sinif daxilindəki x dəyişəninin qiymətini çap edir. Gördüyümüz kimi, private etiketi altında elan etdiyimiz dəyişəni main hissəsində istifadə edə bilməsək də, sinifin öz daxilində istifadə edə bilirik. Beyninizdə, «Niyə private etiketindən istifadə etməliyik?» sualı yarana bilər. Ancaq, bəzi layihələr ola bilər ki, oraada istifadəsinə həqiqətən ehtiyac duyulsun. Bunu da nümunələr yazdıqca görəcəyik. Məsələn, fərz edin ki, bir oyun yazırıq. Oyunçunun istifadəçi adını isə, sinif daxilindəki bir dəyişəndə saxlayırıq. Əgər, oyunçunun adını özündə saxlayan dəyişən public etiketi altında olsa, o zaman, main hissəsində oyunçunun adı dəyişilə bilər. Ancaq, oyunçunun adı yalnız oyuna giriş sinifində təyin olunmalıdır, digər yerlərdən dəyişdirilə bilməməlidir. Buna görə də, burada private tipinə ehtiyac duya bilirik.

```

#include <iostream>

using namespace std;

class sinif{

private:

string gamer_name;

public:

int print_gamer_name(){

this->gamer_name="Shukur";

cout<<this->gamer_name;

}

};

int main(){

```

```

sinif obyekt1;

obyekt1.print_gamer_name();

return 0;

}

```

Gördüyümüz kimi, burada, oyunçunun adı yalnız sınıf daxilində təyin olunur və dəyişən private etiketi altında elan olunduğu üçün, digər yerlərdə dəyişdirilə bilmir. Əsas main funksiyası altında, istifadəçinin adını dəyişməyə cəhd etsək belə dəyişə bilməyəcəyik, çünki, dəyişən private etiketi altındadır və dəyişməyə cəhd etsək proqram işləməyəcək.

Inheritance. Obyektyönlü proqramlaşdırmadakı əsas anlayışlardan biri də, inheritance(miras alma) anlayışdır Bu anlayış vasitəsilə, bir sinifdə elan olunmuş elementləri, digər sinifdə də istifadə edə bilərik. Bunun üçün, sinifləri aşağıdakı şəkildə elan etməliyik:

```

class sinif1{};

```

```

class sinif2 : public sinif1{}

```

Nəticədə birinci sinifin elementlərindən ikinci sinifdə də istifadə edə biləcəyik. Nümunə proqrama baxaq.

```

#include <iostream>

using namespace std;

class sinif1{

public:

int x;

};

class sinif2 : public sinif1{

public:

int funksiya2(){

cout<<x;

```

```

return 0;

}

};

int main(){

sinif2 obyekt2;

obyekt2.x=5;

obyekt2.funksiya2();

return 0;

}

```

Burada, birinci sinif üçün elan etdiyimiz dəyişəndən, ikinci sinifdə də istifadə edə bilirik.

Polimorfizm. Bu anlayış eyni adda funksiyaları müxtəlif şəkildə istifadə edə bilməyə imkan verir. Normalda bir proqramda eyni adda iki funksiya elan edə bilmərik. Nümunə koda baxaq:

```

#include <iostream>

using namespace std;

class sinif{

public:

float sahe(float x, float y){

return x*y;

}

float sahe(float x, float y, float z){

return x*y*z;

}

};

```

```
int main(){  
  
    sinif obyekt1;  
  
    cout<<obyekt1.sahe(5,4)<<"\n"<<obyekt1.sahe(5,4,3);  
  
    return 0;  
  
}
```

Gördüyümüz kimi, burada iki dəfə sahə funksiyası elan etmişik. Proqramın əsas hissəsində funksiyanı istifadə edərkən, verəcəyimiz parametrlərə uyğun olaraq lazımi funksiya istifadə olunacaq. Məsələn, yazdığımız birinci funksiya iki ədəd float tipli parametr alır. Biz main hissəsində funksiyanı istifadə edərkən, parametr olaraq iki ədəd float tipli qiymət versək, o zaman, birinci yazdığımız funksiya istifadə olunacaq. Yox əgər, 3 ədəd float tipli qiymət versək, onda, ikinci yazdığımız funksiya istifadə olunacaq. Ona görə də, proqramda əvvəl 20, sonra isə 60 çap olunur.

C/C++ və HTML

Html, veb səhifələrin skeletini yaratmaq üçün lazım olan dildir. Html-i digər mənbələrdən öyrənmə bilərsiniz. Asan olduğu üçün qısa müddətə öyrənilə bilər. Html-i əvvəlcədən bilirsinizsə, bu mövzunun sizin üçün maraqlı olacağını düşünürük.

Müəyyən yollarla C/C++ və Html-i bir yerdə işlədə bilərik. Aşağıdakı koda baxaq:

```
#include<stdio.h>

int main(){

char *c;

int i;

FILE *fp;

fp=fopen("index.html","a+");

fprintf(fp,"<table border=1>");

for(i=1;i<10;i++){

fprintf(fp,"<tr><td>%d</td></tr>",i);

}

fprintf(fp,"</table>");

system("index.html");

fclose(fp);

return 0;

}
```

Burada, sadə bir əməliyyatı yerinə yetiririk. 1-dən 10- qədər ədədləri for operatoru vasitəsilə html səhifəyə cədvəl şəklində yazırıq. Sonra isə system("index.html"); əmri ilə index.html səhifəsi brauzerdə avtomatik açılır və qarşımıza 1-dən 10-a qədər ədədlərin alt-alta çap olunduğu bir cədvəl çıxır. Linux üzərində isə system("xdg-open index.html"); əmrindən istifadə etməliyik.

Bu üsulla, C/C++ ilə hesabladığımız nəticələri daha yaxşı bir görünüşdə təqdim etməyimiz mümkündür. Eyni zamanda qeyd edək ki, C/C++ çox sürətli dillərdir. Məsələn, veb sahifə üzərində Php dili ilə aparacağımız əməliyyatların sürəti daha aşağı olacaq. Yüksək sürət tələb olunduğu yerlərdə bu üsuldən istifadə edərək əməliyyatları daha tez istifadəçiyə göstərə bilərik. Eyni zamanda, C/C++ ilə bərabər Html/Css/Javascript dillərini də istifadə edərək hesabladığımız nəticələri istifadəçiyə çox gözəl interfeysdə təqdim etməyimiz mümkündür. Məsələn, verilənlər bazasından, veb servislərdən məlumatları oxuyub, nəticəni istifadəçilərə çox gözəl formada təqdim edə bilərik. Məsələn, tək C/C++, Html və Css istifadə edərək, hava proqnozu və valyuta haqqında məlumat verən gözəl bir sayt hazırlamaq mümkündür. Ən sadə halda, Php və ya Asp.net kimi dillərdən istifadə etməyə ehtiyac qalmayacaq. C++ ilə məlumatlar veb servisdən götürüləcək və veb sahifələrə yazılacaq. Eyni zamanda, proqramı elə formada yazmaq olar ki, məlumatlar müəyyən müddətdən bir, məsələn, hər saat və ya hər gün dəyişsin.

Qrafiki interfeysə giriş(C++ Builder)

İndiyə qədər yazdığımız proqramların hamısı konsol üzərində idi. Yəni, istifadəçi yalnız klaviaturadan müəyyən bir simvolu, sözü daxil edərək əməliyyatlar apara bilərdi. Ancaq, qrafiki interfeysdə, istifadəçi hər hansı bir düymənin üzərinə gələrək, düyməni klikləyərək, hər-hansı pəncərədən çıxaraq, düymələrə basaraq bir çox əməliyyatlar apara bilər. Hal-hazırda istifadə etdiyimiz proqramların əksəriyyəti qrafiki interfeysdədir. Word, Excel, Paint, Photoshop və s. kimi proqramlarda düymələrə klik edərək istədiyimiz bir çox əməliyyatı apara bilirik ki, bu da qrafiki interfeys vasitəsilədir.

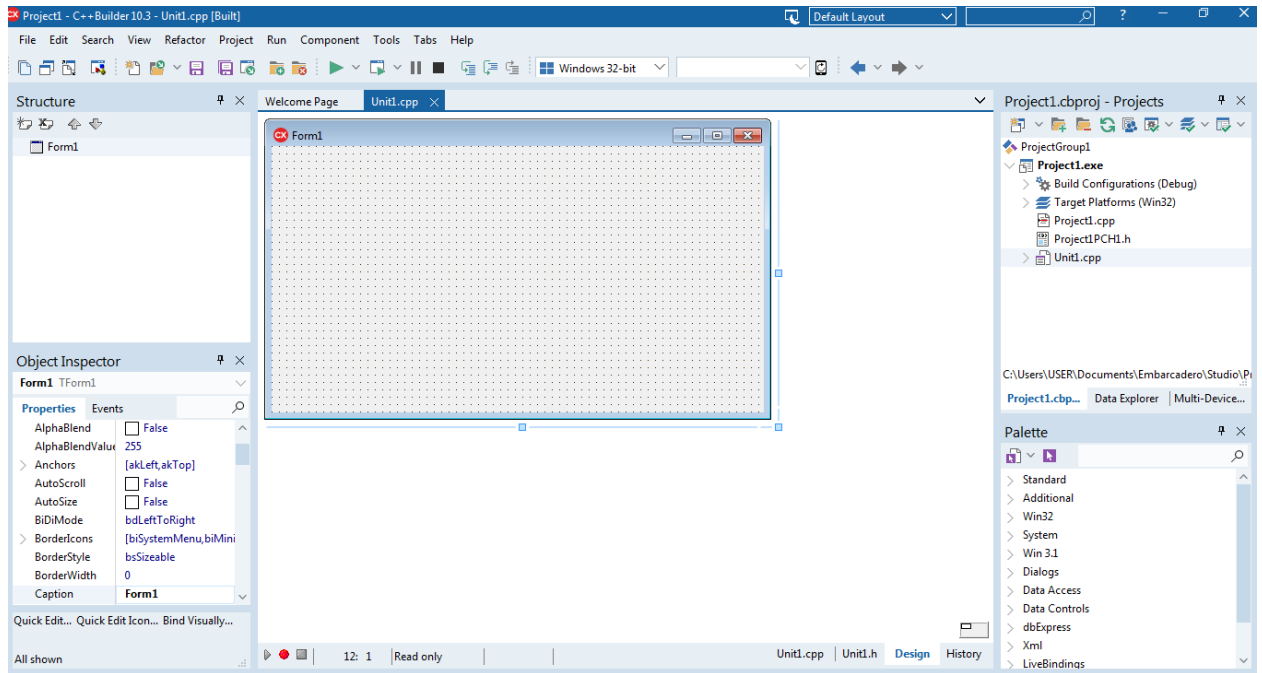
Qrafiki interfeysli proqram yaratmaq üçün istehsal olunmuş bir çox proqramlaşdırma mühiti mövcuddur. C++ ilə istifadə edə biləcəyimiz proqramlaşdırma mühitlərinə misal olaraq C++ Builder, QT və s. göstərə bilərik. Bu mühitlərlə mobil tətbiqlər hazırlamaq da mümkündür. Eyni zamanda yalnızca windows.h kitabxanası ilə də Windows əməliyyat sistemi üçün qrafiki interfeysli proqramlar hazırlamaq mümkündür. Ancaq, bu kitabxana ilə proqramları hazırlamaq çox çətinidir. Bundan əlavə, OpenGL, Graphics.h kimi kitabxanalarla da qrafiki interfeysli proqramlar yaratmaq mümkündür. OpenGL ilə bir sıra oyunlar yaratmaq mümkündür. Eyni zamanda, C++ dəstəkləyən Unreal Engine, CryEngine kimi oyun mühərrikləri ilə də oyunlar hazırlamaq mümkündür.

Sadaladığımız proqramlaşdırma mühitləri, kitabxanalar fərqli olsa da, hər biri eyni məntiqə söykənməkdədir. Hər birində əməliyyatlar baş vermiş müəyyən hadisələr əsasında baş verir. Bu hadisələrə misal olaraq, düyməyə kliklənməsini, düymənin üzərinə gəlinməsini, zamanın keçməsini və s. göstərmək olar.

Biz müxtəlif nümunələr göstərmək üçün C++ Builder proqramlaşdırma mühitindən istifadə edəcəyik. www.embarcadero.com adresindən bu mühiti yükləmək mümkündür. C++ Builder pullu olsa da, ilk olaraq yükləyərək bizə təqdim olunmuş kodla proqramı 1 ay pulsuz işlədə bilərik. Saytda qeydiyyatdan keçib tətbiqi yüklədikdən sonra tələb olunan seriya nömrəsinə elektron poçt ünvanımıza göndərilmiş kodu yaza bilərik. Bununla da proqram bizim üçün 1 aylıq aktiv olmuş olacaq. Proqramın yüklənməsi, quraşdırılması digər windows proqramları kimi çox rahatdır. İlk əvvəl proqramı quraşdırarkən yalnız Windows sistemlərini seçməyimiz kifayətdir. Biz ilk olaraq qrafiki interfeysin məntiqini öyrənmək üçün bu proqramlaşdırma mühitindən istifadə edəcəyik. Öyrəndikdən sonra digər proqramlaşdırma mühitlərindən də istifadə edə bilərsiniz. Qeyd edək

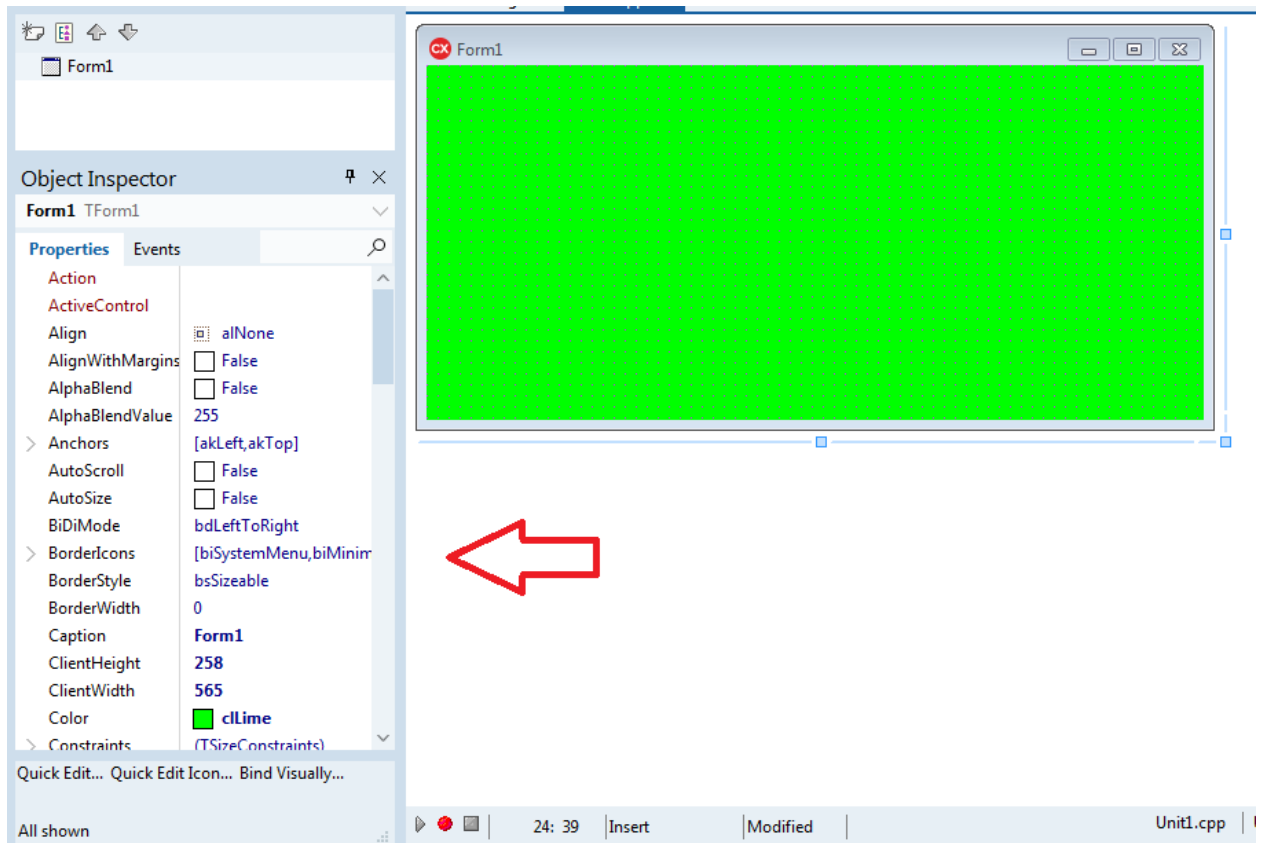
ki, bu mühitlə çox rahatlıqla Android, iOS və Mac sistemləri üçün də tətbiqlər hazırlamaq mümkündür. Qeyd edək, C++ Builder proqramlaşdırma mühitinin daxilində, C/C++ dillərinin özündə olmayan bir çox funksiya və xüsusiyyətlər mövcuddur ki, bunları da irəlində görəcəyik.

Proqramı yükləyib quraşdıqdan sonra proqrama daxil oluruq. Menyudan File->New->Windows VCL Application seçirik. Nəticədə aşağıdakı pəncərə görünür.



Ortada yerləşən pəncərə bizim proqramımızın pəncərəsidir. Proqramımızda yerləşən bütün düymələr, menyular bu pəncərədə yerləşəcək. Burada ümumi bir sinifin olduğunu fərz edə bilərik. Bu sinifdə gördüyümüz pəncərəyə və yerləşdirəcəyimiz elementlərə aid məlumatları özündə saxlayan dəyişənlər mövcuddur. Biz də proqramın gedışı ərzində bu dəyişənlərdən istifadə edə, onlara yeni məlumatlar mənimsədə bilərik.

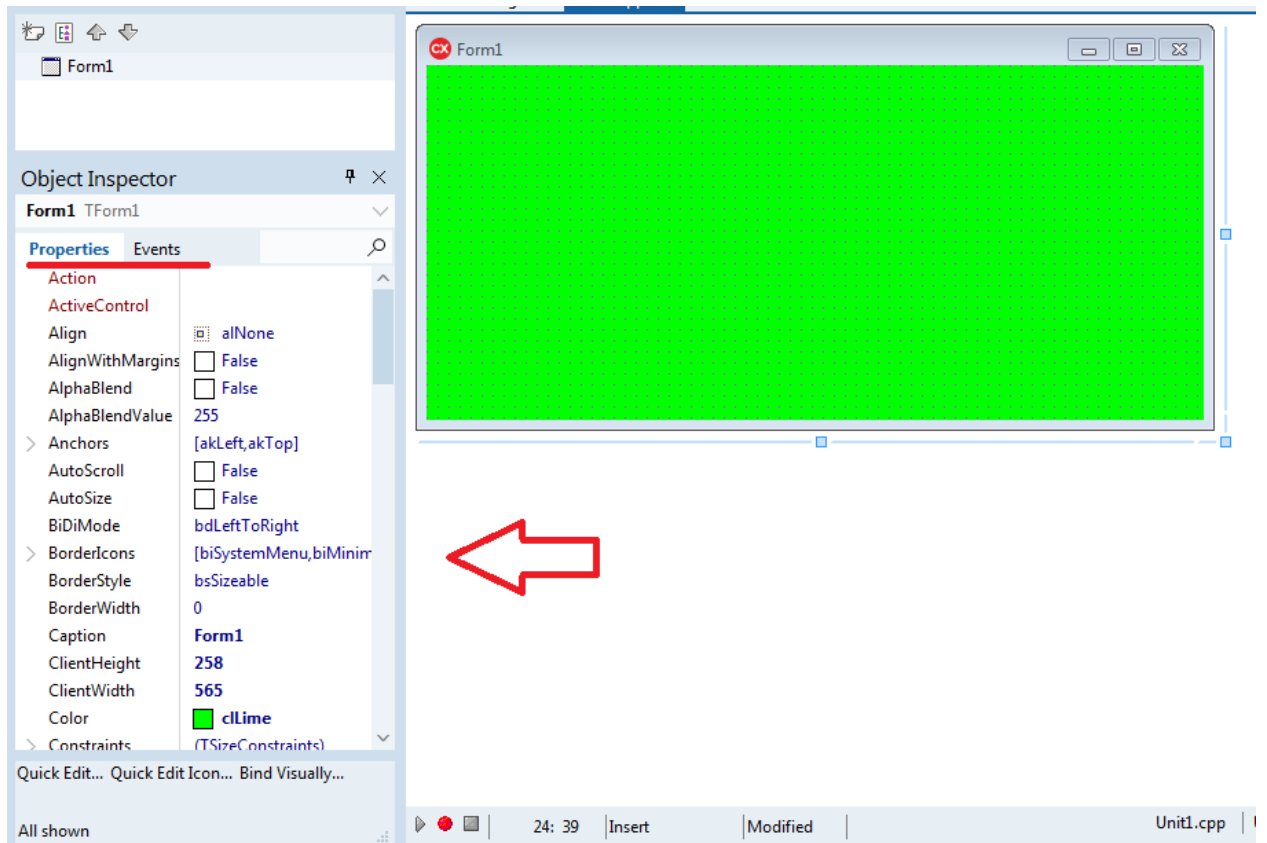
İndi isə yalnız boş pəncərə ilə proqramı işə salaq. Bunun üçün F9 düyməsinə basa bilərik. Nəticədə proqram boş bir pəncərə ilə işə düşəcək. Proqramı işə düşdüyünü gördükdən sonra açılmış proqramı bağlayaq və pəncərənin bəzi xüsusiyyətləri ilə tanış olaq. Ortada yerləşən pəncərənin üzərinə bir dəfə klik edək. Ondan sonra isə sol tərəfdə yerləşən “Object Inspector” pəncərəsinə diqqət edək.



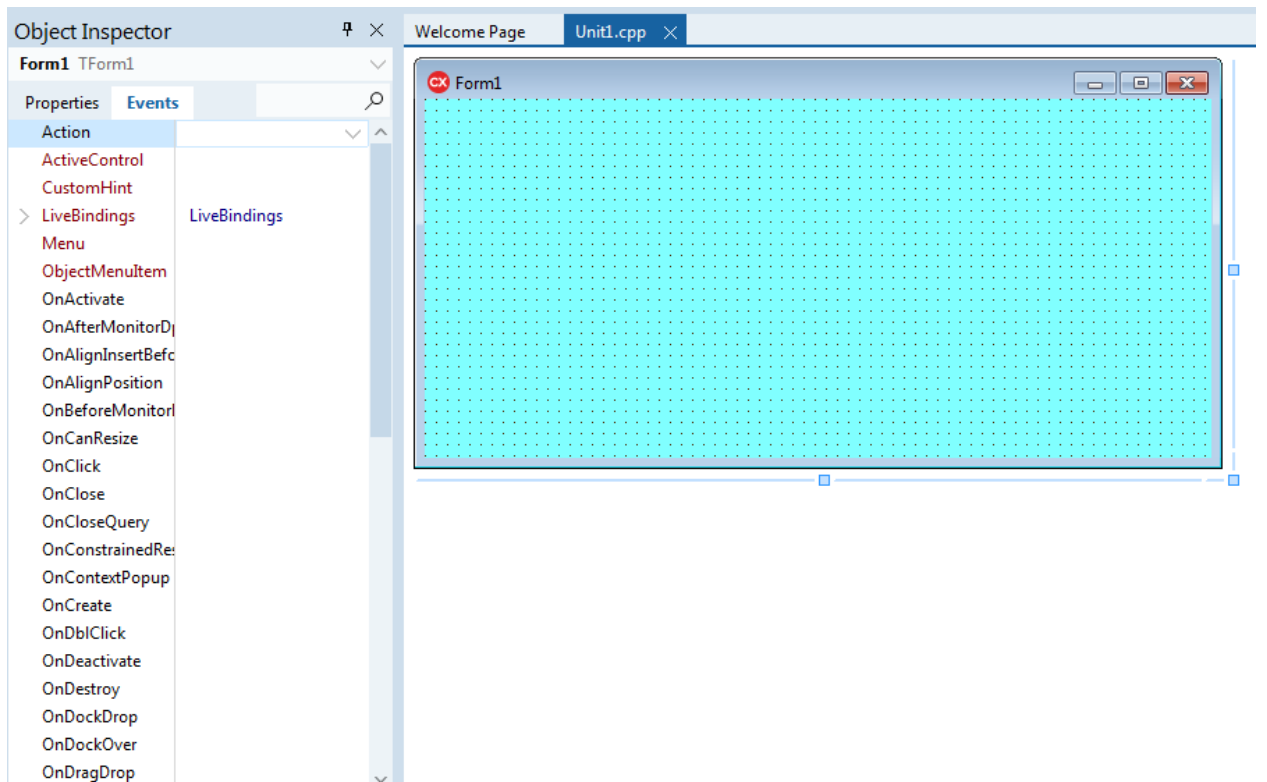
Bu hissədə proqram formuna aid xüsusiyyətlər yer alır. Məsələn, siyahıdan Color xüsusiyyətini taparaq onun qarşısındakı dəyərin üzərinə iki dəfə klik edək və başqa bir rəng seçək. Sonra isə Ok düyməsinə klik edək. Gördüyümüz kimi formanın rəngi dəyişmiş olacaq. İndi isə proqramı F9 düyməsi ilə işlədərək nəticəyə sərbəst proqram şəklində baxa bilərik.

Yuxarıda dediyimiz kimi, bu hissədə proqramın əsas sinifinə aid olan dəyişənlər və onların qiymətləri yer almaqdadır. Buradan onların qiymətini dəyişə bilərik. Burada “width” xüsusiyyəti pəncərənin enini, “height” xüsusiyyəti isə pəncərənin uzunluğunu göstərməkdədir. Bunları dəyişərək proqram pəncərəsinin enini və uzunluğunu dəyişə bilərik.

İndi isə, mövcud hadisələrə baxaq. Dediyimiz kimi, bu interfeysdə bütün əməliyyatlar müəyyən hadisələr əsasında baş verməkdədir. Buna görə də, buradakı hadisələrlə tanış olmalıyıq. Yenidən sol tərəfdə olan “Object Inspector” pəncərəsinə diqqət edək.



Aşağısından qırmızı xətt çəkdiyimiz hissəyə baxaq. Burada iki seçim var. Birinci seçim Properties seçimidir. Hal-hazırda bu seçim aktivdir. Burada program pəncərəsinin xüsusiyyətləri yerləşməkdədir. İkinci seçim isə Events seçimidir. Burada hadisələr yer almaqdadır. Bu seçimə klik edək.



Gördüyümüz kimi, Events seçiminə klik etdikdən sonra mövcud hadisələr açılır. Burada OnClick, OnClose kimi xüsusiyyətlər mövcuddur. İlk öncə OnClick hadisəsindən başlayaq. OnClick hadisəsi elementə klik olunanda müxtəlif hadisələrin baş verməsi üçündür. OnClick sözünün önündəki boş sahəyə iki dəfə klik edək Bundan sonra kod sahəsinin açıldığını görəcəyik.

```
//-----  
void __fastcall TForm1::FormClick(TObject *Sender)  
{  
19  
20 }  
//-----
```

Burada avtomatik olaraq klik hadisəsi üçün bir funksiya yaranır. Bu funksiyanın daxilinə, klik olunanda baş verməsini istədiyimiz hadisələr yazılır.

Yuxarıda da dediyimiz kimi, proqramın pəncərəsinə aid bütün xüsusiyyətlər proqramın əsas sinifində bir dəyişən olaraq yadda saxlanılmışdır və biz onlara baxa, qiymətlərini dəyişə bilərik. İndi isə, daxil olduğumuz klik hadisəsinin funksiyasının daxilində pəncərənin enini və uzunluğunu dəyişək. İlk olaraq proqramın pəncərəsinin adı Form1 olaraq təyin olunmuşdur. Proqramın enini aşağıdakı şəkildə dəyişirik.

```
void __fastcall TForm1::FormClick(TObject *Sender)  
{  
  
    Form1->Width=500;  
  
}
```

Burada Width sözünü böyük hərflə yazdığımıza xüsusi diqqət edək. Çünki, dəyişənin adı böyük hərflə yazılıb. Form1->Width yazaraq pəncərənin uzunluğunu götürmüş oluruq. Bu dəyişəni 1000-ə bərabər edərək isə pəncərənin enini 1000 edirik. İndi isə F9 düyməsinə basaraq proqramı işə salaq. Proqram işə düşdükdən sonra pəncərəyə klik etdikdə pəncərənin eninin artdığını, yəni, 1000-ə bərabər olduğunu görəcəyik. Qrafiki interfeysdə ən əsas məntiq bundan ibarətdir. Müxtəlif hadisələrin funksiyalarına müxtəlif əməliyyatlar yazılaraq lazımi proqram hazırlanır.

Funksiya daxilində bir çox parametri də dəyişə bilərik.

```
void __fastcall TForm1::FormClick(TObject *Sender)
```

```

{

    Form1->Width=1000;

    Form1->Height=1000;

    Form1->Color=clBlue;

}

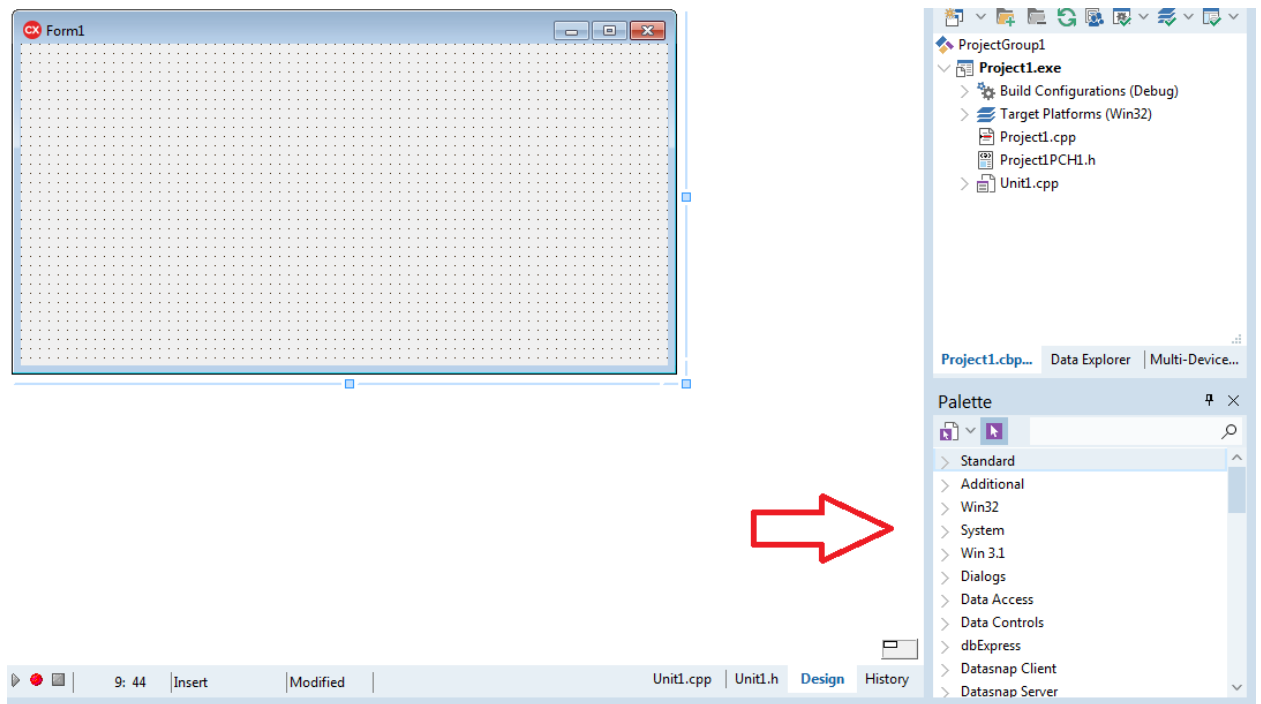
```

Funksiyanın daxilinə yuxarıdakı əməliyyatları yazdıqdan sonra proqramı işə salırıq. Proqram işə düşdükdən sonra proqram pəncərəsinə klik edirik. Nəticədə pəncərənin ölçülərinin və rənginin dəyişdiyini görəcəyik. Burada “clBlue” göy rəngi bildirir.

Yeni tipdə dəyişən yaratmağı xatırlayaq. Əvvəlki mövzularda göstərdiyimiz “enum” ifadəsi ilə fərqli qiymətlər alan dəyişən tipləri yarada bilirdik. Burada da, dəyişən “clBlue” qiymətini alır. Bu da yaradılmış fərqli bir qiymətdir.

Events seçimində olan hadisələrdən biri də OnDblClick hadisəsidir. OnClick hadisəsində olduğu kimi eyni əməliyyatları bu hadisəyə də yaza bilərik. Sadəcə bu seçimin qarşısındakı boş sahəyə iki dəfə klik edib əməliyyatları yazmağımız kifayətdir. Bu hadisədəki əməliyyatlar elementə bir dəfə yox, iki dəfə ardıcıl klik etdikdə baş verir. Yəni, bu düymə bir elementə iki dəfə ardıcıl klik edildikdə əməliyyatların baş verməsi üçündür. Forma üçün OnActivate hadisəsi isə, forma yaradılan anda, yəni proqrama daxil olan anda ilk baş verəcək hadisələr üçündür. OnActivate hadisəsi isə, forma aktiv olduqda baş verəcək hadisələr üçündür. OnClose hadisəsi isə formadan çıxdıqda baş verəcək hadisələr üçündür. Məsələn, proqramdan çıxarkən “Çıxmaq istədiyinizə əminsizmi?” deyə istifadəçidən soruşmağımız mümkündür.

Növbəti olaraq, proqrama düymələrin, yazıların və s. əlavə olunmasına baxaq. Bunun üçün, sağ tərəfdəki “Palette” pəncərəsinə diqqət edək.



Bu pəncərədə proqrama əlavə edə biləcəyimiz elementlər mövcuddur. “Standart” seçiminə klik edək. Bu seçimin altında pəncərəyə əlavə edə biləcəyimiz bir sıra elementlər mövcuddur. Bunlardan biri adi düymə olan “Tbutton” elementidir. Onun üzərinə iki dəfə ardıcıl klik edərək bu elementin əsas pəncərəmizə əlavə olunmasına təmin edə bilərik. Nəticədə düymə əsas pəncərəmizdə olacaq. Onun üzərinə bir dəfə klik edirik. Əsas formamızda olduğu kimi, bu element üçün də xüsusiyyətlər və eventlər mövcuddur. İlk olaraq Properties seçimindən elementin adına baxırıq. İlk olaraq adı Button1 olacaq. Bu ad çox vacibdir, çünki düyməyə bu adla müraciət edəcəyik. Digər vacib bir xüsusiyyəti isə Caption xüsusiyyətidir. Bu xüsusiyyət düymənin üzərinə yazılacaq yazını göstərir. Onu dəyişib “Ok” edə bilərik. İndi isə düymə üçün events seçiminə gələk. Events seçimində OnClick hadisəsinin qarşısındakı boş yerə iki dəfə klik edək və OnClick funksiyasına aşağıdakı kodu yazaq:


```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Button1->Caption="Klik edildi";
}
```

Növbəti olaraq proqramı işə salaq. Proqramı işə saldıqdan sonra düyməyə klik etdikdə, düymənin yazısı dəyişərək "Klik edildi" olacaq.

Növbəti öyrənəcəyimiz element TLabel elementidir. Bu element pəncərə üzərində müxtəlif yazıların görünməsi üçündür. Bu elementin üzərinə iki dəfə klik edək pəncərəyə düşməsinə təmin edək. Bu elementin də ən əsas xüsusiyyətləri Name və Caption xüsusiyyətidir. Name elementin adını, Caption isə üzərindəki yazını göstərir. Name ilk olaraq Label1 olacaq. Caption hissəsinə isə istədiyimiz bir yazını yazı bilərik. Yazının ölçüsünü dəyişmək üçün, Properties seçimində Size altseçiminin qarşısındakı qiyməti dəyişməliyik. Həmin qiyməti 14 edərək yazının ölçüsünün 14 olmasını təmin edə bilərik. Yazının rəngini dəyişmək üçün isə, Fonts seçiminin color altseçimindən istifadə edə bilərik.

Növbəti öyrənəcəyimiz element isə TEdit elementidir. Bu element istifadəçinin proqrama bir məlumat daxil etməsi üçündür. Elementin üzərinə iki dəfə klik edərək pəncərəyə gəlməsinə təmin edək. Proqramı işə saldıqdan sonra elementin daxilindəki yazını dəyişə bildiyimizi görəcəyik. Bu element üçün də ən vacib xüsusiyyətlər Name və Text xüsusiyyətləridir. Name xüsusiyyəti elementin adıdır və ona müraciət üçün istifadə olunur. Elementin adı ilk olaraq Edit1 olaraq təyin olunur. Text xüsusiyyəti isə, elementin daxilindəki yazını götürmək üçündür. İlk olaraq yazı Text1 olaraq qalır. Bu sahəni boşatmaq üçün həmin yazını silə bilərik.

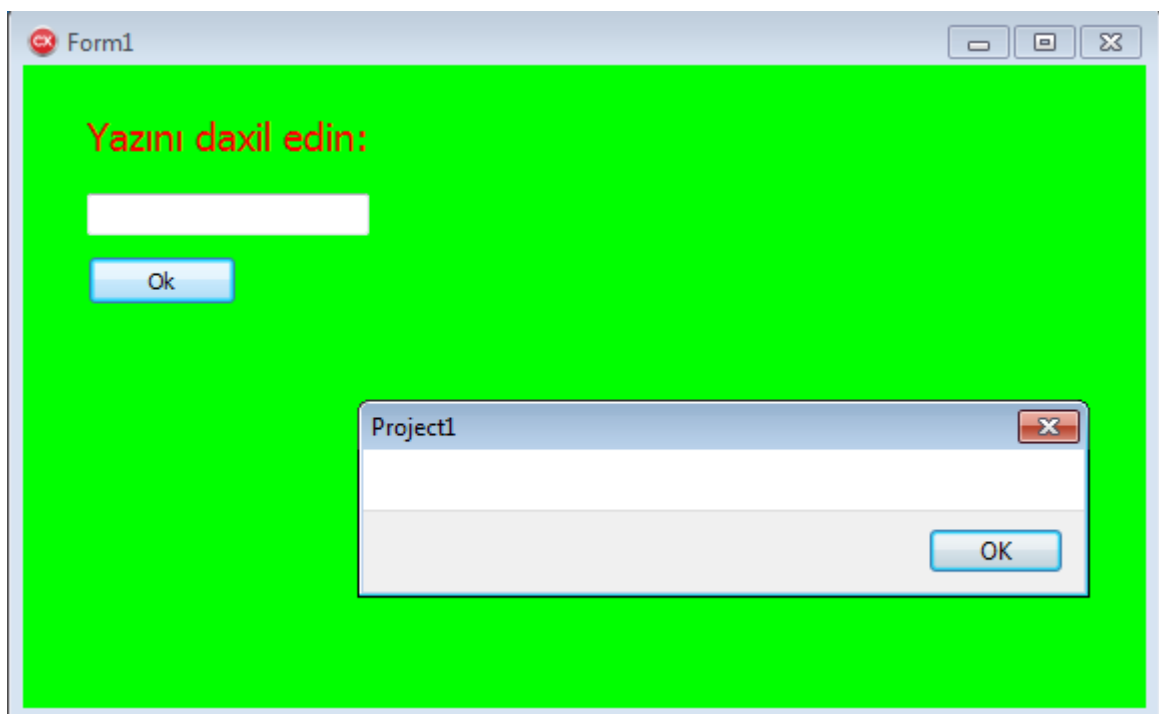
İndi isə, öyrəndiyimiz üç elementdən istifadə edərək aşağıdakı görünüşü hazırlayaq.

The image shows a screenshot of a Windows application window titled "Form1". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main area of the window has a solid red background. In the upper left corner of the red area, the text "Yazını daxil edin:" is displayed in a red, sans-serif font. Below this text is a white rectangular text input field. Underneath the input field is a small, light gray button with the text "Ok" in black. The overall layout is simple and functional, typical of a basic Windows form.

Burada əsas məqsədimiz daxil olunmuş yazını bir pəncərədə ekrana çıxarmaqdır. Bunun üçün düyməmizin OnClick hadisəsinə aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage(Edit1->Text);
}
```

İndi isə proqramı işlədək. Edit elementinə hər hansı bir yazını yazaraq Ok düyməsinə basaq. Nəticədə kiçik bir pəncərədə Edit elementinə yazdığımız yazının çap olunduğunu görəcəyik.



Yazdığımız kodda, Edit1->Text hissəsi ilə Edit elementinin daxilinə yazdığımız yazını götürürük. ShowMessage() funksiyası isə C++ builderin özünə məxsus bir funksiyaadır və bir yazını gördüyümüz pəncərə ilə göstərmək üçündür.

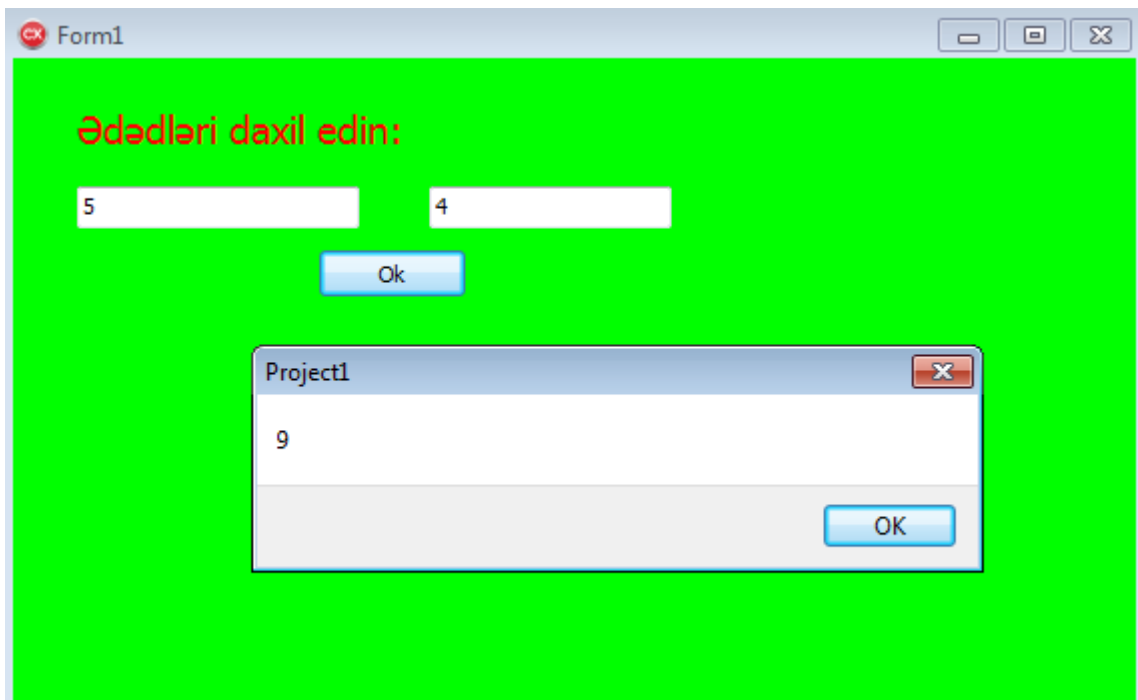
İndi isə bu elementlərlə kiçik bir hesablama proqramı hazırlayaq. Bunun üçün aşağıdakı görünüşü təmin edək.



İndi isə, Ok düyməsinə klik edildikdə, iki ədədin cəmini ekranı çıxaraq. Bunun üçün, Ok düyməsinin OnClick hadisəsi üçün aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    int a,b,c;
    a=Edit1->Text.ToInt();
    b=Edit2->Text.ToInt();
    c=a+b;
    ShowMessage(c);
}
```

Burada a dəyişəni birinci ədəd, b dəyişəni ikinci ədəd, c dəyişəni isə onların cəmidir. Burada ToInt() funksiyası götürdüyümüz ədədi tam tipə çevirmək üçündür. Ədədləri toplamaq üçün onları tam tipə çevirməliyik, çünki, hesablamalar yazı tipi üzərində aparılır. İki ədəd Edit elementimizin qiymətlərini istifadəçidən götürərək cəmini c dəyişəninə yazırıq və ShowMessage() funksiyası ilə nəticəni ekrana çıxarıyıq. İndi isə proqramı işə salaq. İki ədəd daxil etdikdən sonra Ok düyməsinə klik etdikdə, onların cəminin çıxdığını görəcəyik.



Bu proqramı biraz daha genişləndirə bilərik. Bunun üçün aşağıdakı görünüşü təmin edək:



Burada yazdığımız ədədlər üzərində 4 müxtəlif əməliyyat aparacağıq. Bunun üçün Topla düyməsinin OnClick hadisəsinə aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    int a,b,c;
```

```
a=Edit1->Text.ToInt();  
b=Edit2->Text.ToInt();  
c=a+b;  
ShowMessage(c);  
}
```

Çıx düyməsinin OnClick hadisəsi üçün:

```
void __fastcall TForm1::Button2Click(TObject *Sender)  
{  
    int a,b,c;  
    a=Edit1->Text.ToInt();  
    b=Edit2->Text.ToInt();  
    c=a-b;  
    ShowMessage(c);  
}
```

Vur düyməsinin OnClick hadisəsi üçün:

```
int a,b,c;  
a=Edit1->Text.ToInt();  
b=Edit2->Text.ToInt();  
c=a*b;  
ShowMessage(c);
```

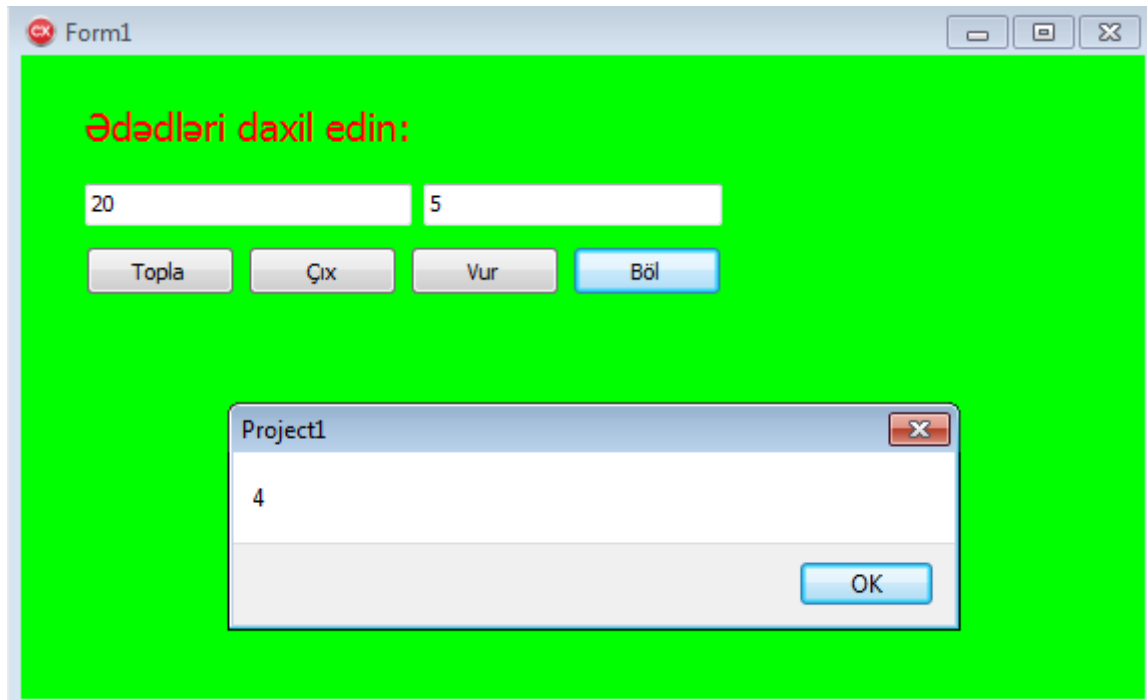
Böl düyməsinin OnClick hadisəsi üçün:

```
int a,b,c;  
a=Edit1->Text.ToInt();  
b=Edit2->Text.ToInt();
```

```
c=a/b;
```

```
ShowMessage(c);
```

İndi isə proqramı işə salaq. Nəticədə müxtəlif düymələrlə toplama, çıxma, vurma və bölmə əməliyyatlarını yerinə yetirə bilərik.



Bütün əməliyyatlar doğru bir şəkildə yerinə yetiriləcək. Ancaq, bölmə üçün 19 və 4 yazıb bölmə əməliyyatı aparmağa çalışsaq, nəticədə 4 çıxacaq. Çünki, biz int tipindən istifadə etmişik. Tam dəqiq nəticə üçün, int tipi yerinə float və ya double tipindən istifadə edə bilərik.

```
void __fastcall TForm1::Button4Click(TObject *Sender)
```

```
{
```

```
double a,b,c;
```

```
a=Edit1->Text.ToInt();
```

```
b=Edit2->Text.ToInt();
```

```
c=a/b;
```

```
ShowMessage(c);
```

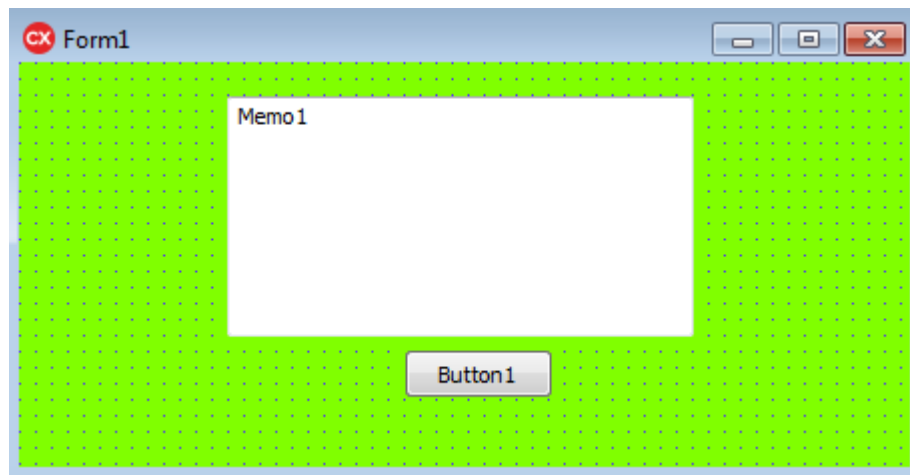
```
}
```

Bu şəkildə yazıb proqramı işə saldıqda, tam dəqiq nəticələr əldə biləcəyik. Eyni zamanda 0-a bölmənin olmadığını da yada salmaq. Buna görə də, bölmə zamanı ikinci ədədin 0-a bərabər olub-olmadığını yoxlaya bilərik. Əgər, ikinci ədəd 0 olarsa, hesablamaqdan imtina edə bilərik. Bunun üçün, Böl düyməsinin OnClick hadisəsinin kodlarını aşağıdakı şəkildə dəyişə bilərik.

```
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    if(Edit2->Text=="0"){
        ShowMessage("0-a bölmək olmaz.");
    }
    else{
        double a,b,c;
        a=Edit1->Text.ToInt();
        b=Edit2->Text.ToInt();
        c=a/b;
        ShowMessage(c);
    }
}
```

Burada şərtlə ikinci Edit elementinin qiymətinin 0-a bərabər olub-olmadığını yoxlayırıq. Əgər qiyməti 0 olarsa, hesablama aparmayaraq bunu istifadəçiyə bildiririk. Nəticədə daha dəqiq bir proqram yazmış oluruq.

Növbəti öyrənəcəyimiz element, nisbətən böyük mətnlərin proqram daxilində yazılması üçün nəzərdə tutulan Memo elementidir. Palette pəncərəsinin Standart seçiminin altında yerləşən TMemo adlı elementin üzərinə iki dəfə klik edək və onu əsas pəncərəyə yerləşdirək. Onunla bərabər bir düymə də yerləşdirək.



İstifadə etdiyimiz Memo elementinin ən əsas xüsusiyyətlərindən biri Name xüsusiyyətidir. Name xüsusiyyətindən elementin adı yazılır və proqram ərzində ona bu adla müraciət olunur. İkinci əsas xüsusiyyəti isə Lines xüsusiyyətidir. Bu xüsusiyyətdə, bu elementin daxilinə yazılacaq yazılar yer alır. Proqram ərzində bu elementə yazılan yazıları götürmək üçün, bu xüsusiyyətdən istifadə edəcəyik.

Yuxarıda formaya düyməni və Memo elementini yerləşdirdikdən sonra, düymənin OnClick hadisəsinə aşağıdakı kodu yazmaq və proqramı işlədək.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage(Memo1->Lines->GetText());
}
```

Burada **Memo1->Lines->GetText()** yazaraq elementə daxil edilmiş yazıları götürmüş oluruq. ShowMessage() ilə də bu yazıları bir pəncərədə ekrana çıxarıyıq. Nəticədə, düyməyə basıldıqda, Memoya daxil olunan yazılar kiçik bir pəncərədə ekrana çıxacaq.

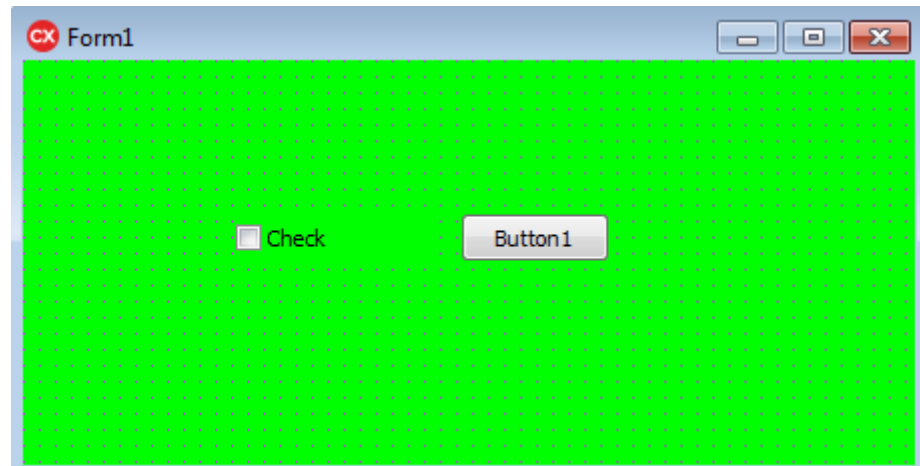
C dilində öyrəndiyimiz fayl funksiyalarından əlavə, sırf C++ Builderin daxilində Məlumatı fayla yazmaq üçün də SaveToFile() funksiyası mövcuddur. Yuxarıdakı nümunədə düymənin OnClick hadisəsinə aşağıdakı kodu yazmaq:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Memo1->Lines->SaveToFile("newfile.txt");
}
```


}

Proqramı işlədib düyməyə klik etdikdən sonra, proqramın işlədiyi qovluqda, "newfile.txt" adında faylın yarandığını və Memo elementinə daxil etdiyimiz yazıların bu fayla yazıldığını görəcəyik.

Növbəti öyrənəcəyimiz element, Palette pəncərəsinin Standart seçiminin Checkbox elementidir. Bu elementi taparaq pəncərəyə yerləşdirək. Onunla bərabər bir düymə də pəncərəyə yerləşdirək.



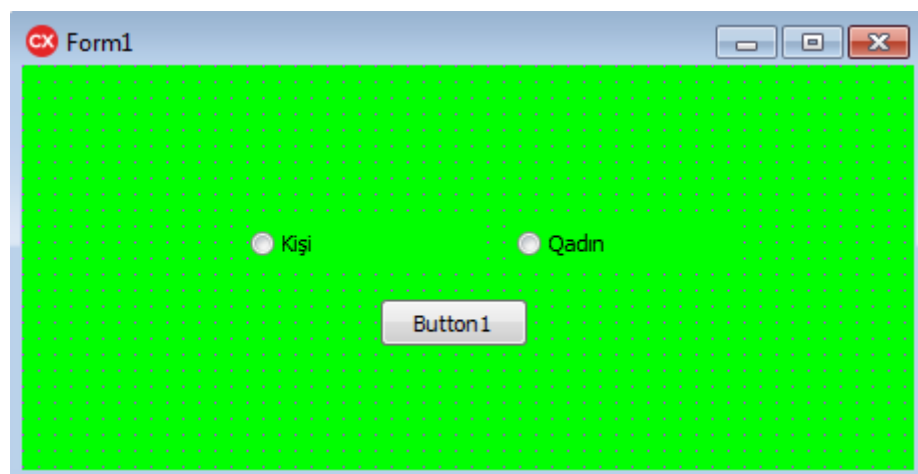
Checkbox elementini veb saytlarda da tez-tez görürük. Hər-hansı bir seçimin seçilib-seçilməməsini göstərir. Əsas xüsusiyyətlərindən biri name xüsusiyyətidir. Name xüsusiyyəti elementin adını göstərir. İkinci xüsusiyyət isə Caption xüsusiyyətidir. Bu xüsusiyyət seçimin yanında yazılacaq yazını göstərir. Elementin digər bir xüsusiyyəti isə checked xüsusiyyətidir. Element seçili olduqda checked xüsusiyyətinin qiyməti true olur, seçili olmadıqda isə false olur. İndi isə, düymənin OnClick hadisəsinə aşağıdakı kodu yazaq.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if(CheckBox1->Checked==true){
        ShowMessage("Seçilmişdir.");
    }
    else{
        ShowMessage("Seçilməmişdir.");
    }
}
```

```
}
```

Proqramı işə salaq. Nəticədə, element seçildikdə Seçilmişdir yazısı, seçilmədikdə isə seçilməmişdir yazısı ekrana çıxacaq.

Növbəti öyrənəcəyimiz element RadioButton elementidir. Bu element, bir neçə variantdan yalnız birinin seçilməli olduğu vəziyyətlərdə istifadə olunur. Məsələn, cinsi seçərkən, 5 test variantından birini seçərkən və s. istifadə olunur. Bu elementin də, əsas xüsusiyyətləri Name, Checked və Caption xüsusiyyətləridir. Name xüsusiyyəti elementin adını, Checked xüsusiyyəti elementin seçilib-seçilmədiyini, Caption xüsusiyyəti isə seçimin yanındakı yazını göstərir. İki ədəd RadioButton Elementindən istifadə edərək aşağıdakı pəncərəni yaradaq.



Burada istifadəçinin öz cinsini seçməsi üçün iki seçim yerləşdirdik. Düymənin OnClick hadisəsinə aşağıdakı kodu yazaq.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if(RadioButton1->Checked==true){
        ShowMessage("Kişi.");
    }
    else if(RadioButton2->Checked==true){
        ShowMessage("Qadın.");
    }
    else{
```

```

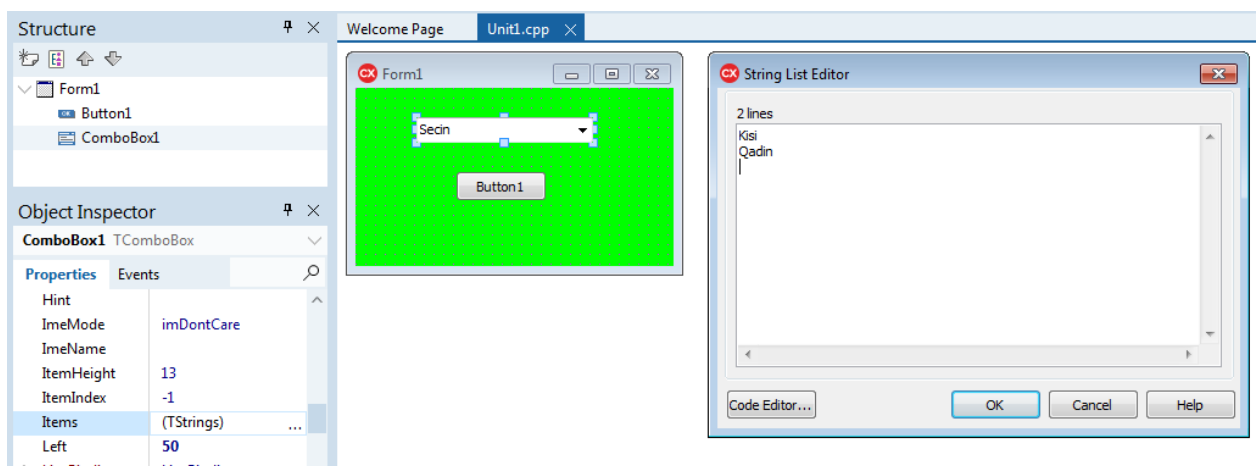
ShowMessage("Seçilməmişdir.");
}

}

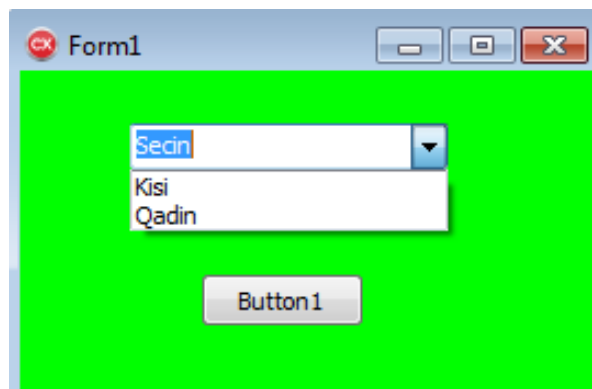
```

Nəticədə seçimə uyğun yazı ekrana çıxacaq. Heç biri seçilmədikdə isə, "Seçilməmişdir." Yazılır.

Öyrənəcəyimiz elementlərdən biri də, Standart seçiminin altında yerləşən ComboBox elementidir. Məsələn, saytlarda ölkələrin siyahısını göstərərək, bir çox halda bu elementdən istifadə olunur. Elementin əsas xüsusiyyətləri Name, Text, Items və ItemIndex xüsusiyyətləridir. Name elementin adını, Text ilk olaraq elementdə göstəriləcək yazını, Items seçimləri ItemIndex isə seçimin index nömrəsini göstərir. Heç nə seçilməyən halda index nömrəsi -1 olur, ilk seçimdə isə 0 olur. Burada da sayma 0-dan başlayır.



Ekrana bir ədəd ComboBox və düymə yerləşdiririk. ComboBoxun Items xüsusiyyətinin qarşısındakı Strings yazısına klik edirik. Klik etdikdən sonra bir pəncərə açılır. O pəncərədə seçimləri alt-alta yazırıq. Burada seçimlərə Kişi və qadın yazmışıq. Nəticədə proqram işə düşdükdən sonra ComboBox açılarkən Kişi və qadın seçimləri görünür.



İndi isə, düymənin OnClick hadisəsinə aşağıdakı kodu yazaq:

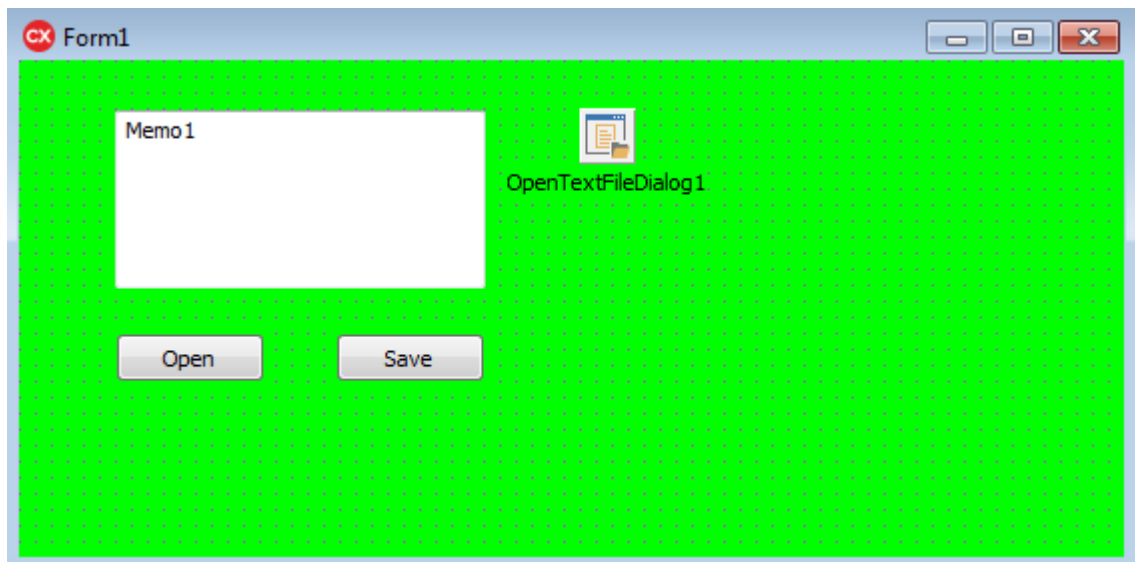
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if(ComboBox1->ItemIndex==-1){
        ShowMessage("Seçilməmişdir.");
    }
    else if(ComboBox1->ItemIndex==0){
        ShowMessage("Kişi seçilmişdir.");
    }
    else if(ComboBox1->ItemIndex==1){
        ShowMessage("Qadın seçilmişdir.");
    }
}
```

Nəticədə, seçimə uyğun nəticə çıxacaq. -1 qiyməti seçimin olmamağını göstərir. Seçim 0-dan başladığı üçün, 0 olduqda "Kişi seçilmişdir." yazılır, 1 olduqda isə "Qadın seçilmişdir." yazılır.

Həmin siyahıdakı Listbox elementi də, ComboBox elementinə çox oxşarlıq göstərməkdədir. ComboBox elementini ListBox elementi ilə əvəzləyərək də bir sıra əməliyyatlar apara bilərsiniz.

Komputerdə yerləşən hər-hansı bir mətn faylını açıb yazıları Memo elementinin daxilində göstərə, dəyişiklik edə, və onu yadda saxlaya bilərik. İlk öncə masaüstündə bir text faylı yaradaq və içinə müəyyən yazılar yazaq.

Sonra isə, proqram pəncərəsinə bir ədəd Memo, iki ədəd düymə və bir ədəd Dialogs seçiminin altında yerləşən OpenFileDialog elementini yerləşdirək.



Bu pəncərəni hazırladıqdan sonra, Open düyməsinin OnClick hadisəsinə aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    OpenTextFileDialog1->Execute();
    Memo1->Lines->LoadFromFile(OpenTextFileDialog1->FileName);
}
```

Proqramı işə salaq. Open düyməsinə basdıqda, **OpenTextFileDialog1->Execute();** sətiri vasitəsilə ilə element işə düşəcək və komputerdən bir faylın seçilməsi mümkün olacaq. Text tipli yaratdığımız bir fayl seçildikdən sonra, ikinci sətir vasitəsilə text faylındakı yazılar Memo elementimizə yüklənmiş olacaq. **OpenTextFileDialog** elementi əsas olaraq komputerdən bir faylı seçmək üçün istifadə olunur. OpenTextFileDialog1->FileName hissəsi, seçilmiş faylın adını tam ünvanı ilə götürmək üçün istifadə olunur.

İndi isə Save düyməsinin OnClick hadisəsinə aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Memo1->Lines->SaveToFile(OpenTextFileDialog1->FileName);
}
```

Nəticədə, Save düyməsinə basıldıqda, Memo elementinə köçürülmüş yazıda edilmiş dəyişikliklər olduğu kimi həmin faylda yadda saxlanacaq. Bu xüsusiyyəti ShowMessage() funksiyası ilə ekrana çıxarıb aldığı qiymətə baxa bilərsiniz.

Memo elementinə yazdığımız yazını başqa bir fayl adı ilə də yadda saxlamağımız mümkündür. Bunun üçün ekrana bir ədəd üzərinə Save as yazacağımız düymə, bir ədəd də Dialogs seçiminin altında yerləşən SaveTextFileDialog elementini yerləşdirək və Save as düyməsinin OnClick hadisəsinə aşağıdakı kodu yazaq:

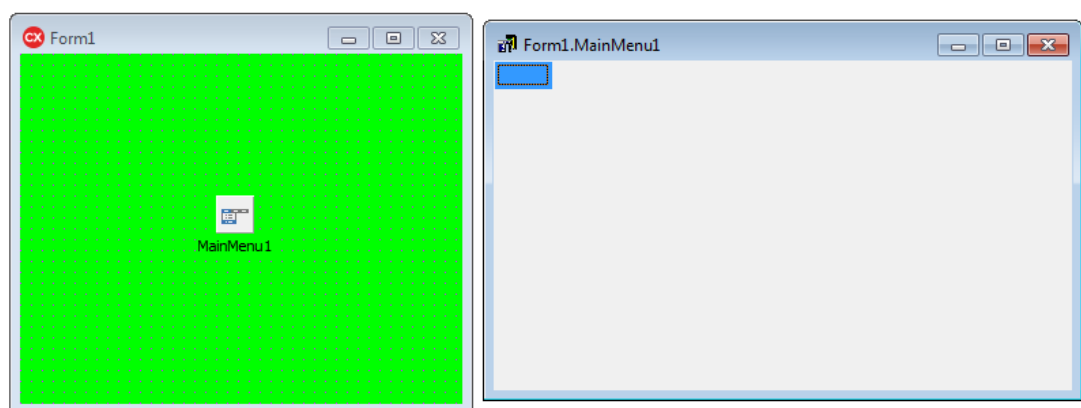
```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    SaveTextFileDialog1->Execute();

    Memo1->Lines->SaveToFile(SaveTextFileDialog1->FileName);
}
```

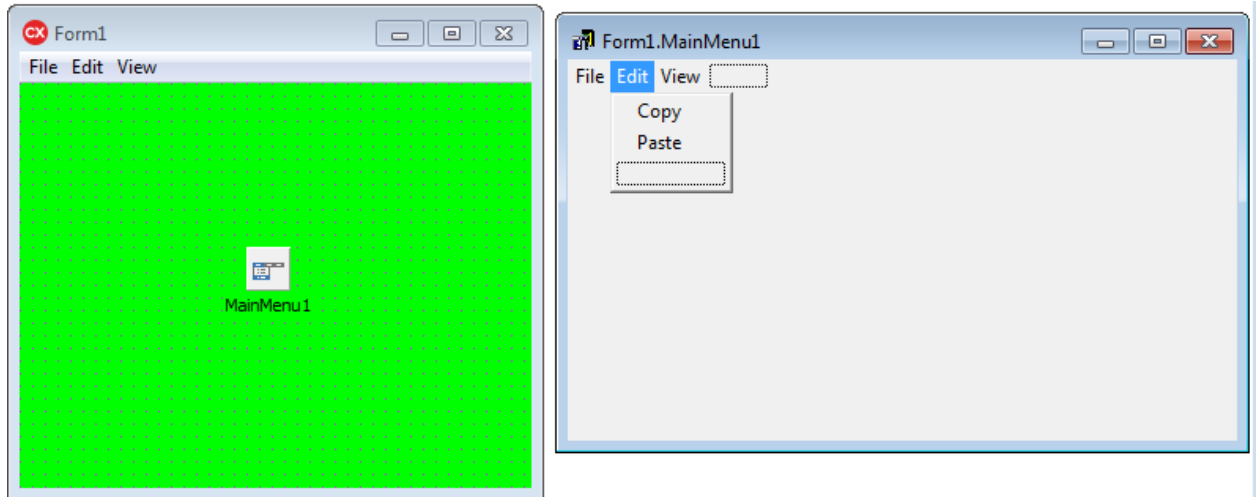
Nəticədə, Save as düyməsinə klik etdikdə, açılacaq pəncərəyə faylın_adı.txt yazırıq və yadda saxlayırıq. Nəticədə, seçdiyimiz qovluqda, Memo elementində yerləşən yazıların köçürüldüyü və seçdiyimiz adda olan bir text faylının yarandığını görəcəyik.

Hazırlayacağımız proqramlarda, Menyular xüsusi əhəmiyyət daşıyır. Menyular vasitəsilə istifadəçi bir çox əməliyyatları yerinə yetirir. Standart seçimi altında yerləşən MainMenu elementi vasitəsilə, proqrama əsas menyular əlavə etməyimiz mümkündür. Əsas menyu dedikdə, bir çox proqramların yuxarı hissəsində yerləşən File, Edit və s. kimi menyuları nəzərdə tuturuq.

MainMenu elementini əsas pəncərəyə yerləşdirək və üzərinə iki dəfə klik edək. Nəticədə aşağıdakı pəncərə açılacaq.



Açılan pəncərədə menyu hissəsinin üzərikə klik edək və Caption xüsusiyyətinin qarşısına menyunun adını yazaq, məsələn File. Bu adı yazıb enter dedikdən sonra, bu bölmənin alt menyusu açılır. Eyni qayda ilə bir neçə menyu yaradıb altbölmələrini yarada bilərik.



Bu şəkildə menyuları yaratmış oluruq. Hər-hansı bir menyuya klik edildikdə hər-hansı bir hadisənin baş verməsini istəyiriksə, onda, həmin menyunun üzərinə klik edirik və OnClick hadisəsinə istədiyimiz kodları yazırıq. Məsələn:

```
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
    ShowMessage("Ok");
}
```

Əsas menyudan əlavə popup menyular da mövcuddur. Bu menyular proqram daxilində hər-hansı bir düymənin üzərində sağ düyməni basdıqda açılır. Popup menyulardan istifadə etmək üçün, Standart seçiminin altında yerləşən Poup elementini formaya yerləşdirmək lazımdır. Ondan sonra isə elementə menyuları yazı bilərik. Menyu bölmələrinin OnClick hadisəsinə istədiyimiz hadisələri yazmağımız mümkündür. Elementə menyuları yazdıqdan sonra elementi başqa bir elementə aid etməliyik. Məsələn, bu menyunun proqram daxilində formanın üzərində sağ düyməni basdıqda açılmağını istəyiriksə, Formanın üzərinə klik edib, PopupMenu xüsusiyyətindən menyunu seçməyimiz kifayətdir. Müxtəlif elementlərin üzərində sağ düyməni klik etdikdə fərqli menyuların açılmasını istəyiriksə, onda, bir neçə popup menu istifadə edib, onların hərəsini uyğun elementə yerləşdirməliyik.

Şəkillər proqramlara xüsusi görünüş qatmaqdadır. Bəzi proqramlar isə birbaşa şəkillərin özləri ilə işləyir. Eyni zamanda, oyun proqramlaşdırmasında da şəkillərdən geniş istifadə oluna bilər. Additional seçiminin altında yerləşən Image elementi vasitəsilə pəncərəyə şəkillər əlavə etməyimiz mümkündür. Bunun üçün, həmin elementi pəncərəyə yerləşdirək. Bundan sonra həmin elementin Picture xüsusiyyətinə klik edək. Açılacaq pəncərədə Load düyməsinə klik edib istədiyimiz bir şəkli seçək və sonda Ok düyməsinə klik edək. Nəticədə şəkil əsas proqram pəncərəsinə yerləşmiş olacaq.



Beləliklə, şəkil pəncərəyə yerləşmiş olacaq. İlk olaraq, şəklin proqramda normal düşməsi üçün, şəkli yerləşdirdikdən sonra Image elementinin enini və uzunluğunu, şəklin eni və uzunluğuna bərabər etməliyik.

Bizim üçün maraqlı olacaq bir çox elementlər də, Palette pəncərəsinin Win32 seçiminin altında yerləşir. Burada, Windowsda yerləşən bir çox standart elementlər mövcuddur. İlk öncə Animate elementindən başlayaq. Animate elementini proqrama yerləşdirərək CommonAVI xüsusiyyətindən bir animasiya seçək. Burada, faylın silinməsinə, kopyalanmasını və s. hadisələri animasiya şəklində göstərən seçimlər mövcuddur. Onlardan birini seçdikdən sonra elementin Active xüsusiyyətini true edək və proqramı işə salaq. Nəticədə, element daxilində animasiya olduğunu görəcəyik.

Bu seçimdəki TrackBar elementi, Windowsda səsi artırmaq və ya azaltmaq üçün olan düyməyə oxşardır. Min qiyməti, alınacaq minimum dəyəri, Max qiyməti isə alınacaq maksimum dəyəri göstərir. Elementin Orientation xüsusiyyəti isə, elementin Horizontal, yoxsa Vertical istiqamətdə dayanacağını təyin edir. Position xüsusiyyəti isə, elementin üzərindəki oxun yerini göstərir.

ProgressBar elementi isə, müəyyən bir əməliyyatın miqdarını göstərmək üçün istifadə olunur. Bundan əvvəlki elementə oxşar xüsusiyyətləri mövcuddur. Proqramı bu elementlə bərabər bir Timer də yerləşdirək. Timerin interval xüsusiyyətinin qiymətini 1 edək və Timerin OnTimer hadisəsinə aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    ProgressBar1->Position+=1;
}
```

Kodu yazdıqdan sonra proqramı işə salaq, Nəticədə elementin Pozisiyası hər bir millisaniyədə bir vahid artaraq kiçik bir animasiya yaratmış olacaq.

Öyrənəcəyimiz digər elementlərdən biri də, Palette pəncərəsinin System seçiminin altında yerləşən Timer elementidir. Bu elementin əsas funksiyası, eyni əməliyyatları bir müddətdən bir, məsələn 3 saniyədən bir yerinə yetirmək üçün istifadə olunur. Elementin əsas xüsusiyyətləri Name, Interval və Enabled xüsusiyyətləridir. Name xüsusiyyəti timerin adını bildirir. Interval xüsusiyyəti əməliyyatın neçə müddətdən bir yerinə yetiriləcəyini göstərir, millisaniyə ilə yazılır. Məsələn, 1000 yazıldıqda 1 saniyə nəzərdə tutulur. Enabled xüsusiyyəti isə Timerin işləyib-isləməyəcəyini göstərir. Bu xüsusiyyətin qarşısında seçilmiş işarəsi olduqda timer işləyir, olmadıqda isə işləmir.

Timer elementini tapaq və pəncərəyə yerləşdirək. Bu element, proqram işləyərkən görünmür, sadəcə, onun uyğun hadisəsinə yazılmış əməliyyatlar yerinə yetirilir. Yerləşdirdiyimiz timerin Interval xüsusiyyətinə 3000 yazaq. OnTimer hadisəsinə isə ShowMessage("Ok"); yazaq. Proqramı işə saldıqdan sonra, hər 3 saniyədən bir ekranda kiçik pəncərədə "Ok" yazıldığını görəcəyik.

İndi isə əsas pəncərəyə bir ədəd timer və label yerləşdirək. İlk öncə Labelin qiymətini 0, Timerin Intervalını isə 1000 edək və Timerin OnTimer hadisəsinə aşağıdakı kodu yazaq:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    Label1->Caption=Label1->Caption.ToInt()+1;
}
```

}

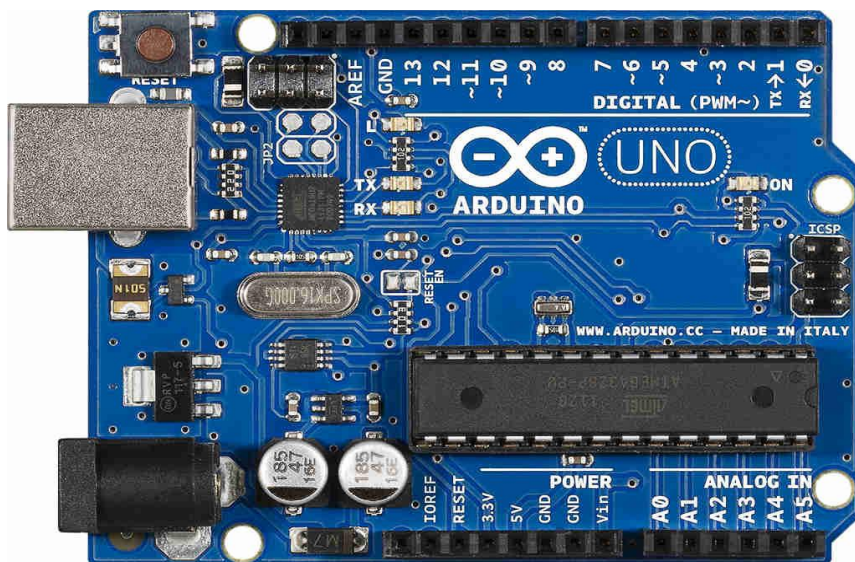
Proqramı işə saldıqdan sonra Labeldəki qiymət hər bir saniyədə bir vahid artmış olacaq. Burada **Label1->Caption.ToInt()** yazmağımızın səbəbi, toplama üçün String tipli ifadəni int tipinə çevirməli olduğumuza görədir. Bu funksiya da C++ Builderə məxsus bir funksiyadır.

Yuxarıda göstərdiyimiz əsas elementlərdən əlavə, C++ Builderə məxsus bir sıra elementlər mövcuddur ki, onları Palette pəncərəsindən seçərək istifadə edə bilərsiniz. Palette pəncərəsində Standart seçimindən sonra Additional seçimi mövcuddur. Bu seçimin altında bir çox element yerləşir. Bu elementlər, Standart seçimin altındakı elementlərə oxşarlıq göstərməkdədir, ancaq bir sıra əlavə xüsusiyyətləri də mövcuddur. O elementlərin də, digər elementlərə oxşar bir çox Eventləri mövcuddur ki, onun vasitəsilə də bu elementlərdən istifadə edərək bir çox əməliyyatları yerinə yetirməyimiz mümkündür. Additional bölməsinin elementlərini də bir-bir yoxlayıb, müxtəlif əməliyyatlar aparamığınız məsləhət görülür. Həmçinin, digər seçimlərin altında da müxtəlif əməliyyatları yerinə yetirən bir çox element mövcuddur.

Cihaz proqramlaşdırılması

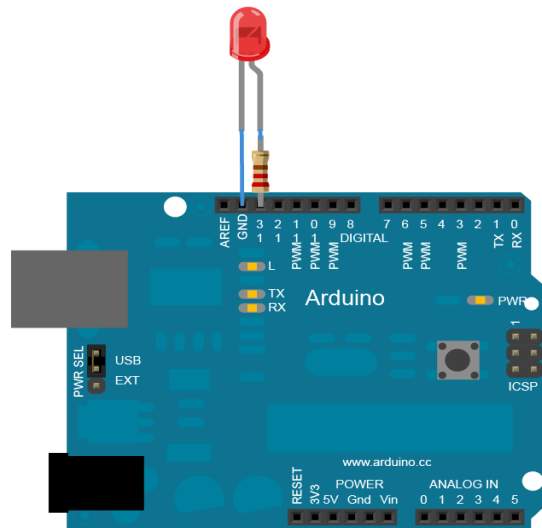
C/C++ vasitəsilə, kontrollerləri proqramlaşdıraraq müxtəlif qurğular, hətta robotlar hazırlamaq mümkündür. Bu kitabda bu mövzunun qısa olaraq yazılmasında əsas məqsəd, C/C++ vasitəsilə belə bir şeyin mümkün olduğunu da göstərməkdir.

C/C++ istifadə edərək bir sıra qurğular hazırlaya biləcəyimiz proqramlaşdırma platformaları mövcuddur. Bunlara misal olaraq Arduino, Raspberry Pi və s. göstərmək olaraq. Bu platformalar üzərində bir sıra prosesləri avtomatlaşdırmaq, qurğular hazırlamaq, robot proqramlaşdırmaq mümkündür. Bunları etmək üçün Arduino platformasına, Arduino üçün proqramlaşdırma mühitinə bir sıra elektronika elementlərinə ehtiyacımız var. Arduionun bir sıra növləri mövcuddur. İlk olaraq bizə Arduino UNO kifayət edəcəkdir. Ölkə daxilindən, eyni zamanda, alışveriş saytlarından Arduionu əldə etməyiniz mümkündür. Bundan əlavə, Led, Rezisitor, Fotorezistor kimi bir sıra elektronika elementlərinə ehtiyacımız var. Bundan əlavə, Arduino kodlarını yazacağımız proqramlaşdırma mühitinə ehtiyacımız var. Bu proqramlaşdırma mühitini Arduionun öz saytıdan (Arduino.cc) pulsuz şəkildə yükləməyimiz mümkündür.



Arduionun üzərində şəkildə yuxarıda hissədə 2-13 olaraq işarələnmiş rəqəmsal giriş/çıxışlar mövcuddur. Onların qiyməti 1 olduqda elektrik gəlir, 0 olduqda isə həmin sahədə elektrik olmur. GND olaraq yazılmış girişə isə, elementin mənfi ayağı qoşulur. Eyni zamanda, aşağı hissədə göstərildiyi kimi, A0-A5 analoq giriş/çıxışlar da mövcuddur. Bu mövcuddur. Bunlar vasitəsilə müxtəlif gərginliklər vermək mümkündür. Eyni zamanda, element üzərində bir neçə ədəd də GND girişi, 5 Volt və 3.3 Volt çıxışları da mövcuddur.

İndi isə Arduino ilə, qoşulmuş lampanı bir saniyədən bir söndürüb yandıran proqram yazaq. Bunun üçün, Arduino ilə aşağıdakı dövrəni quraq:



Ledin mənfi ayağını GND çıxışına, digər ayağını isə 13 nömrəli çıxışa yazaq. Sonra isə, aşağıdakı kodu yazaq və Arduinoya yükləyək.

```
void setup() {  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(13, HIGH);  
  delay(1000);  
  digitalWrite(13, LOW);  
  delay(1000);  
}
```

Yazdığımız proqramı Arduinoya yüklədikdən sonra, Arduinoya qoşulmuş Led bir saniyədən bir sönüb yanacaq. İndi isə kodun izahın keçək.

Burada iki əsas funksiya var. Birinci funksiya setup() funksiyasıdır. Bu funksiya daxilində, proqram işə düşdükdən sonra olacaq ilk tənzimləmələr aparılır.

İkinci funksiya isə loop() funksiyasıdır. Bu funksiyanın daxilində isə, daimi olaraq əməliyyatlar aparılır.

Loop funksiyasının daxilində digitalWrite() və delay() funksiyalarından istifadə etmişik. digitalWrite() funksiyası seçilmiş nömrəli pinə 0 və ya 1 verir. Delay() funksiyası isə gözləmə üçündür, 1000 parametrini aldıqda 1000 millisaniyə, yəni, 1 saniyə gözləyir.

DigitalWrite() funksiyası iki parametr alır. Birinci parametr pinin nömrəsidir. İkinci parametr isə HIGH və ya LOW qiymətlərini alır. HIGH qiyməti olduqda çıxışa elektrik verilir, LOW qiyməti olduqda isə elektrik verilmir, yəni 0 qiymətini alır.

İlk başda setup() funksiyasının daxilində yazdığımız pinMode(13, OUTPUT); funksiyasında isə, 13 nömrəli pinin çıxış kimi istifadə olunacağını bildirir.

Proqramlaşdırma məsələləri

Proqramlaşdırma məntiqini daha yaxşı anlamaq üçün, sadədən mürəkkəbə bir çox məsələləri etmək çox yaxşı olar. Məsələlərin həllinə baxmazdan əvvəl özünüz düşünməyiniz, məsələni özünüz həll etməyiniz daha yaxşı olar. Göstərilmiş məsələlərin həlli mümkün qədər sadə izahla yazılmağa çalışılsa da, bəzi məsələlərin nisbətən çətin olduğunu nəzərə alaraq, məsələlərin həlli üçün videolu izahlara da baxmağınız məsləhət görülür. Bir sıra məsələləri və onların həllini aşağıda göstərək.

1) 1-dən 100-ə qədər ədədlərin cəmini tapın. Bunun üçün 1-dən 100-ə qədər ədədləri sıralayıb onları digər bir dəyişənin üzərinə gəlməliyik. Bu dəyişənimiz isə say dəyişənidir ki, /onun ilk qiyməti də 0-a bərabərdir. Çünki sıralamada ilk qiymət 1 olacaq və həmin 0-ın üzərinə gəlinəcək. Sonra sıralamadakı növbəti ədəd say dəyişəninə yeni qiymətinin üzərinə gəlinəcək, onda, say 3 alınacaq. Bu qayda ilə 1-dən 100-ə qədər bütün ədədlər toplanmış olacaq.

```
#include <stdio.h>
```

```
int main(){
```

```
int i;
```

```

int say=0;

for(i=1;i<=100;i++){

say=say+i;

}

printf("%d",say);

getchar();

return 0;

}

```

2) 1-dən 10-a qədər ədədlərin hasilini tapın. Bu məsələ də yuxarıdakına oxşardır. Ancaq, burada say dəyişəninin ilk qiyməti 1 olacaq. Çünki növbəti ədəd 1-ə vurulacaq. Əgər say dəyişəninin ilk qiyməti 0 olarsa, onda, növbəti ədədlər də 0-a vurulacaq və nəticə tapılmayacaq. Əgər 1-dən 100-ə qədər ədədlərin hasilini tapmaq istəsək, onda, say dəyişəninin float və ya double tipindən istifadə etməyimiz daha məqsədəuyğun olar. Çünki 1-dən 100-ə qədər ədədlərin hasilində kifayət qədər böyük bir ədəd alınır.

```

#include <stdio.h>

int main(){

int i;

int say=1;

for(i=1;i<=10;i++){

say=say*i;

}

printf("%d",say);

getchar();

return 0;

}

```

3) Daxil edilmiş ədədin faktorialını tapın. Daxil edilmiş ədədin faktorialını tapmaq üçün, 1-dən həmin ədədə qədər ədədləri bir-birinə vurmaliyiq.

```
#include <stdio.h>

int main(){

int i,daxil;

int say=1;

scanf("%d",&daxil);

for(i=1;i<=daxil;i++){

say=say*i;

}

printf("%d",say);

getchar();

return 0;

}
```

4) Sadə ədədlərin tapılması. Əvvəlcə alqoritmi təyin edək. Sadə ədəd yalnız özünə və 1-ə tam bölünən ədədlərə deyilir. Məsələn, 17 sadə ədəddir. Çünki, 17 yalnız özünə və 1-ə tam bölünür. Bir ədədin sadə olub-olmadığını tapmaq üçün, onu özünə qədər bütün ədədlərə bir-bir bölməli və bölünüb-bölünmədiyini yoxlamaliyiq.

```
#include <stdio.h>

int main(){

int i,sade,daxil;

scanf("%d",&daxil);

sade=1;

for(i=2;i<daxil;i++){

if(daxil%i==0){
```

```

sade=0;

break;

}

}

if(sade==0){

printf("Eded sade deyil.");

}

else if(sade==1){

printf("Eded sadedir.");

}

getchar();

return 0;

}

```

İlk olaraq ədədin sadə olduğunu fərz edirik və $sade=1$; olaraq qeyd edirik. Sonra isə 2-dən tapmalı olduğumuz ədəddən 1 vahid kiçik ədədə kimi bütün ədədləri yoxlayırıq. Əgər ədədimiz onlardan hər-hansı birinə bölünürsə, $sade$ dəyişənini 0 edirik və dövrəni sıdırırıq. Çünki, əgər ədəd 2-dən sonra hər-hansı bir ədədə bölünübsə deməli sadə deyil. Sonda $sade$ dəyişəninin 0-a bərabər olub-olmadığını yoxlayırıq. Əgər, dəyişən 0-a bərabərdirsə, deməli, ədəd sadə deyil. Əks halda isə ədəd sadədir.

5) Ədədin əks yazılışının tapılması. 6541 ədədinin əks yazılışını aşağıdakı şəkildə tapa bilərik:

$$((1*10+4)*10+5)*10+6;$$

Burada, ədədin əks yazılışı üçün ilk öncə onun son ədədini 10-a vururuq və üzərinə ondan bir vahid əvvəlki ədədi gəlirik. Burada son ədədimiz 1 olduğu üçün, $1*10+4=14$ alınır. Sonra isə alınmış bu ədədi yenə 10-a vurub üzərinə 5 gəlirik. Nəticədə, $14*10+5=145$ alınır. Sonra yenə bu ədədi bir daha 10-a vuraraq üzərinə daha əvvəlki ədədi, yəni 6-ı gəlirik və $145*10+6=1456$ ədədini alırıq. Nəticədə

ədədin əks yazılışını tapmış oluruq. İlk öncə daxil edilmiş dörd rəqəmli ədədin əks yazılışının tapılması üçün proqramı yazaq.

```
#include <stdio.h>

int main(){

int daxil,eded1,eded2,eded3,eded4;

daxil=6541;

eded4=daxil%10;

daxil=daxil/10;

eded3=daxil%10;

daxil=daxil/10;

eded2=daxil%10;

daxil=daxil/10;

eded1=daxil;

printf("%d",((eded4*10+eded3)*10+eded2)*10+eded1);

getchar();

return 0;

}
```

Proqramda 5 ədəd dəyişənimiz var, daxil dəyişəni ədədi təmsil edir, eded4 ədədin son rəqəmini, eded3 ədədin üçüncü rəqəmini, eded2 ədədin ikinci rəqəmini, eded1 isə ədədin ilk rəqəmini təmsil edir. Ədədin 10-a bölünməsindən alınan qalıq, ədədin son rəqəmidir. Buna görə də, $eded4 = daxil \% 10$; yazaraq ədədin son rəqəmini tapmış oluruq. Növbəti sətirdə isə $daxil = daxil / 10$; yazaraq ədədin 10-a bölünməsindən alınan tam hissəni götürürük və daxil dəyişənimizin yeni qiyməti 654 olmuş olur. Bunu etməyimizə səbəb ədədin digər rəqəmlərini götürməyimizi asanlaşdırmaqdır. Növbəti olaraq, bu ədədin 10-a bölünməsindən alınan qalığı tapırıq ki, bu isə 4 rəqəmidir. Bu isə, əsas ədədimizin üçüncü rəqəmidir. Sonra yenə ədədi 10-a bölərək tam hissəni tapırıq ki, bu da 65 rəqəmidir. Növbəti sətirlərdə də yenə eyni qayda ilə ədədin digər rəqəmlərini

tapmış oluruq. Sonda isə, $((\text{eded4} * 10 + \text{eded3}) * 10 + \text{eded2}) * 10 + \text{eded1}$ ifadəsi ilə, ədədin əks yazılışını tapmış oluruq.

Aşağıdakı proqramda isə, tək dörd rəqəmli ədədlər üçün yox, bütün ədədlər üçün əks yazılışı tapa bilərik.

```
#include <stdio.h>

int main(){

int daxil,qaliq,eks;

scanf("%d",&daxil);

while(daxil!=0){

qaliq=daxil%10;

daxil=daxil/10;

eks=eks*10+qaliq;

}

printf("%d",eks);

getchar();

return 0;

}
```

Burada bir dövrə başladırıq və dövrədəki şərtimiz ədədin 0-dan fərqli olmasıdır. Dövrədə ilk olaraq qaliq dəyişəni ilə ədədin son rəqəmini götürürük. Sonra isə, ədədin növbəti rəqəmlərinin də tapılması üçün $\text{daxil} = \text{daxil} / 10$; yazaraq ədədin 10-a bölünməsindən alınan tam hissəni götürürük, yəni, ədədin son rəqəmini itiririk. Sonra isə, $\text{eks} = \text{eks} * 10 + \text{qaliq}$; yazaraq son hissənin əks yazılışını tapırıq. Bundan sonra isə dövrə yenidən başlayır. Proqram daxilində $\text{daxil} = \text{daxil} / 10$; ifadəsi 0 qiymətini alana qədər dövrə davam edir və hesablama aparılır. Nəticədə ədədin əks forması çap olunur.

6) Ədədin daxilində 5 rəqəminin olub-olmadığının yoxlanması.

```
#include <stdio.h>

int main(){
```

```

int daxil,qaliq;

int var=0;

scanf("%d",&daxil);

while(daxil!=0){

qaliq=daxil%10;

if(qaliq==5){

var=1;

}

daxil=daxil/10;

}

if(var==1){

printf("Daxilinde 5 var.");

}

else{

printf("Daxilinde 5 yoxdur.");

}

getchar();

return 0;

}

```

Əvvəlki proqramlara oxşar olaraq burada da ədədin rəqəmlərini bir-bir yoxlayaraq onların hər-hansı birinin 5-ə bərabər olub-olmamasını yoxlayırıq. Əgər, ədədin rəqəmlərindən hər-hansı biri 5-ə bölünürsə, o zaman, var dəyişənini 1-ə bərabər edirik. Sonda isə var dəyişəninin 1-ə bərabər olub-olmamasını yoxlayırıq. Əgər var dəyişəni 1-ə bərabərdirsə. deməli, ədədin rəqəmlərindən ən azı biri 5-ə bərabərdir.

7) 1-dən 10-a qədər ədədlər üçün vurma cədvəlinin yazılması.

```
#include <stdio.h>

int main(){

int i,j;

for(i=1;i<=10;i++){

for(j=1;j<=10;j++){

printf("%d * %d=%d \n", i, j, i*j);

}

printf("-----\n");

}

return 0;

}
```

Burada iç-içə iki ədəd dövrədən istifadə edirik. Hər iki dövrədə ədədlər 1-dən 10-a qədər sıralanır. Birinci dövrədə ilk olaraq i dəyişənin qiyməti 1 olur. Sonra isə ikinci dövrəyə keçilir. İkinci dövrədə j dəyişənin qiyməti 1-dən 10-a qədər sıralanır və i dəyişənin ilk qiymətinə, yəni 1-ə vurulur. Nəticədə 1 üçün vurma cədvəli yazılmış olur. İçəridəki dövrə tamamlandıqdan sonra, yenidən birinci dövrə qaydılır və i dəyişənin yeni qiyməti 2 olur. Yenidən daxiləki dövrə keçilir və daxiləki dövrədə j dəyişənin qiyməti yenidən 1-dən 10-a qədər sıralanır və i dəyişənin qiymətinə, yəni 2-ə vurulur. Bu qayda ilə, i dəyişəni 1-dən 10-a qədər sıralanır və hər qiyməti 1-dən 10-a qədər sıralanmış j dəyişəninə vurulur. Nəticədə, 1-dən 10-a qədər vurma cədvəli yazılmış olur.

8) Fibonaççi ədədlərinin tapılması. Fibonaççi ədədləri aşağıdakı ardıcılıqla gedir:

0,1,1,2,3,5,8,13,21,34,55

Bu ardıcılıqda, hər növbəti ədəd, əvvəlki iki ədədin cəminə bərabər olur. Aşağıdakı proqramla müəyyən qiymətə qədər olan Fibonaççi ardıcılığı tapılmış olur.

```
#include <stdio.h>
```

```

int main(){
int eded1,eded2,yadda;

eded1=0;

eded2=1;

printf("%d, %d, ",eded1,eded2);

while(eded1<100){

printf("%d,",eded1+eded2);

yadda=eded2;

eded2=eded1+eded2;

eded1=yadda;

}

return 0;

}

```

Proqramın izahı üçün izahlı videoya baxmağınız məsləhət görülür.

9) Çoxluqdakı ədədlərin cəminin tapılması. Burada, çoxluqdakı ədədlər bir-bir sıralanaraq, çoxluqdakı ədədlərin cəmi tapılır.

```

#include <stdio.h>

int main(){

int a[9],i,cem;

cem=0;

printf("Ededleri daxil edin:\n");

for(i=0;i<9;i++){

scanf("%d",&a[i]);

}

for(i=0;i<9;i++){

```

```

cem=cem+a[i];

}

printf("\n\n\n%d",cem);

return 0;

}

```

10) Çoxluqdakı maksimum ədədin tapılması.

```

#include <stdio.h>

int main(){

int a[9],i,max;

printf("Ededleri daxil edin:\n");

for(i=0;i<9;i++){

scanf("%d",&a[i]);

}

max=a[0];

for(i=1;i<9;i++){

if(a[i]>max){

max=a[i];

}

}

printf("\n\n\n%d",max);

return 0;

}

```

11) Çoxluqdakı minimum ədədin tapılması.

```

#include <stdio.h>

```

```

int main(){
int a[9],i,min;

printf("Ededleri daxil edin:\n");

for(i=0;i<9;i++){

scanf("%d",&a[i]);

}

min=a[0];

for(i=1;i<9;i++){

if(a[i]<min){

min=a[i];

}

}

printf("\n\n\n%d",min);

return 0;

}

```

12) 100-dən 300-ə qədər ədədlərin arasında, rəqəmləri bir-birindən fərqli olan ədədləri və onların sayını çap edin.

```

#include <stdio.h>

int main(){

int i,say;

int reqem1, reqem2, reqem3;

for(i=100;i<=300;i++){

reqem1=i%10;

reqem2=(i/10)%10;

reqem3=i/100;

```

```

if(reqem1!=reqem2 && reqem1!=reqem3 && reqem2!=reqem3){

    printf("%d\n",i);

    say++;

}

}

printf("\n\n\n%d",say);

return 0;

}

```

13) Bir sinifdə tələbələrin aldığı qiymətlər proqramda daxil olunmuşdur. Sinifin ortalamasını, ən yüksək və aşağı qiymətini tapın.

```

#include <stdio.h>

int main(){

int a[9],i,max,min,orta,cem;

printf("Ededleri daxil edin:\n");

for(i=0;i<9;i++){

scanf("%d",&a[i]);

}

max=a[0];

min=a[0];

for(i=0;i<9;i++){

if(a[i]>max){

max=a[i];

}

if(a[i]<min){

min=a[i];

```



```

}

cem=cem+a[i];

}

printf("\n\n\n Max:%d Min: %d Ortalama: %d",max,min,cem/9);

return 0;

}

```

14) Daxil edilmiş mətndə, nöqtələri vergüllə əvəzləyən proqram yazın.

```

#include <algorithm>

#include <string>

#include <iostream>

int main() {

std::string s = "example string";

std::replace( s.begin(), s.end(), 'x', 'y'); // replace all 'x' to 'y'

std::cout<<s;

return 0;

}

```

15) Daxil edilmiş iki ədəd aralığındakı bütün ədədləri bir-birinə vuran ikiparametrli funksiya yazın.

```

#include <stdio.h>

int funksiya(int x,int y){

int hasil,i;

hasil=1;

for(i=x;i<=y;i++){

hasil=hasil*i;

}

```

```

return hasil;

}

int main() {

printf("%d",funksiya(3,4));

return 0;

}

```

16) Mətn daxilində “a” hərfinin olub-olmadığını yoxlayan program yazın.

```

#include <iostream>

int main(){

std::string a;

int i,var,say;

var=0;

a="Metn daxilinde a herfi axtarilacaq.";

for(i=0;i<a.length();i++){

if(a.substr(i,1)=="a"){

var=1;

break;

}

}

if(var==1){

std::cout<<"Metn daxilinde a herfi var.";

}

else{

std::cout<<"Metn daxilinde a herfi yoxdur.";

```

```
}
```

```
return 0;
```

```
}
```

17) Mətn daxilində olan "a" hərflərinin sayını tapın.

```
#include <iostream>
```

```
int main(){
```

```
    std::string a;
```

```
    int i,say;
```

```
    say=0;
```

```
    a="Metn daxilinde a herfi axtarilacaq.";
```

```
    for(i=0;i<a.length();i++){
```

```
        if(a.substr(i,1)=="a"){
```

```
            say++;
```

```
        }
```

```
    }
```

```
    std::cout<<say;
```

```
    return 0;
```

```
}
```

Linuxa giriş

Linux nüvəsi üzərində bir çox əməliyyat sistemləri hazırlanmışdır. Bunlara misal olaraq Ubuntu, Fedora, Kali Linux kimi sistemləri göstərmək olar. Bu sistemlər Windows sistemindən fərqlənməkdədir. Windows üzərində istifadə etdiyimiz proqramlar, Linux əsaslı sistemlərdə işləməməkdədir. Ancaq yenə də, Windows proqramlarına alternativ olaraq Linux əsaslı sistemlərdə də oxşar proqramlar mövcuddur. Məsələn, Notepad yerinə gedit, Ms Office yerinə Libre Office, Photoshop yerinə Gimp istifadə etmək olar. Linux əsaslı bir çox sistemin müsbət cəhətlərini aşağıdakı kimi sıralaya bilərik:

- 1) Linux əsaslı sistemlərdən bir çoxu pulsuzdur, məsələn Ubuntu. Heç bir pul ödəmədən Ubuntu əməliyyat sistemini öz komputerinizə quraşdırıla bilərsiniz.
- 2) Daha təhlükəsizdir, virus demək olar ki yoxdur. Windows üzərində yazılmış viruslar burada işləməyəcək, çünki dediyimiz kimi Windows proqramları birbaşa Linux əsaslı sistemlərdə işləməməkdədir. Linux sistemi üçün viruslar yazılsa belə, Linux sistemlərində hər əməliyyatı yerinə yetirərkən şifrə tələb olunduğuna görə, virus lazımi əməliyyatları yerinə yetirə bilməyəcək. Windowsda flashkartı bir komputərə taxıb başqasına taxdıqda belə virus düşə bilər. Ancaq, Linuxda bunlar yoxdur.
- 3) İqtisadi cəhətdən çox sərfəlidir. Dediyimiz kimi, həm sistemlərin özü pulsuzdur, həm də, virus falan olmadığına görə antivirus proqramları almağa da ehtiyac duymursunuz.
- 4) OpenSource sistemdir. Bu o deməkdir ki, əməliyyat sisteminin nüvəsində dəyişikliklər edib, sistemə yeni funksionallıqlar əlavə edə bilərik. Qeyd edək ki, Linux nüvəsi də C dili ilə yazılıb buna görə də, əməliyyat sistemi üzərində bir yenilik etmək üçün, C dilinə yüksək dərəcədə hakim olmaq lazımdır.

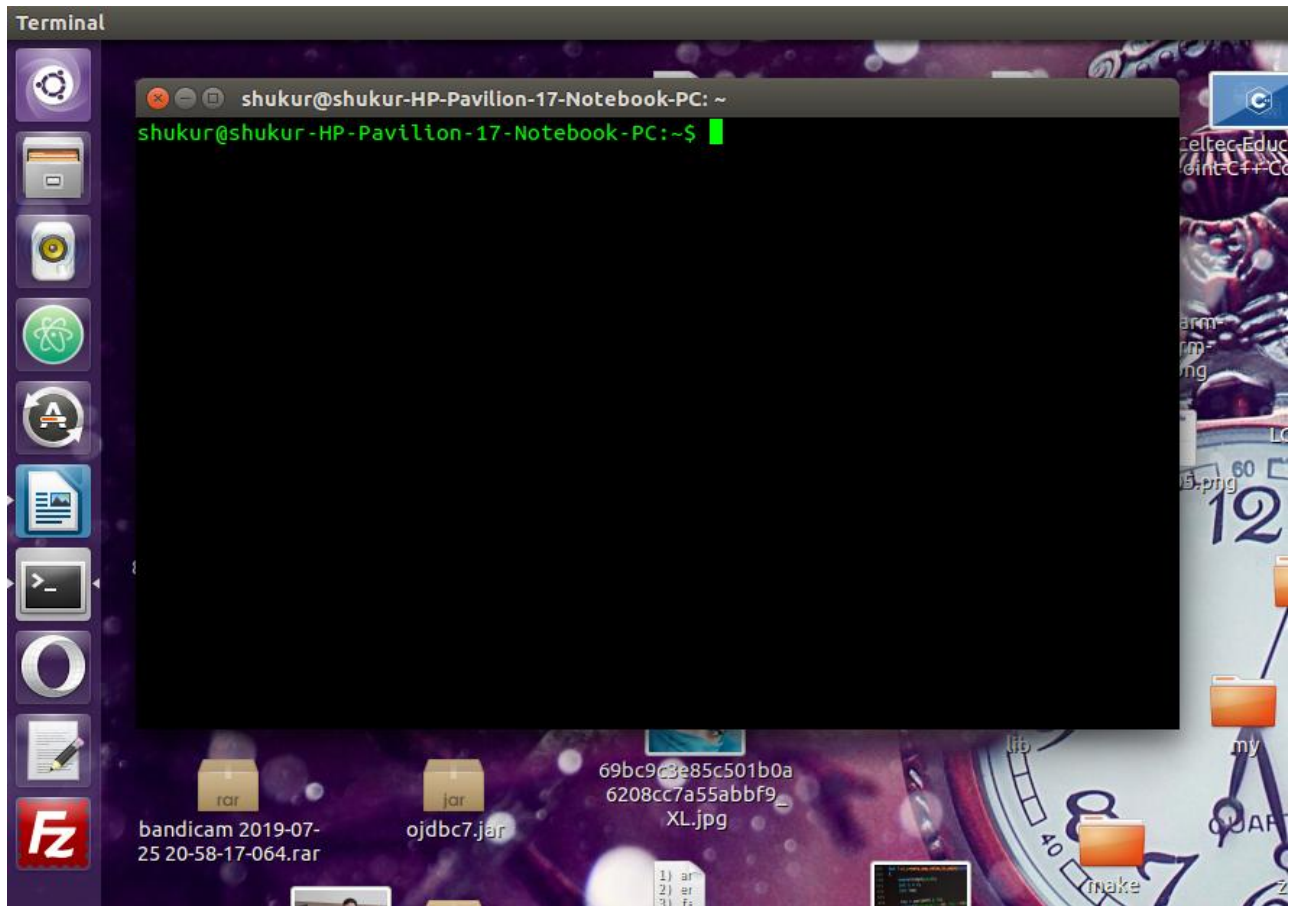
Müsbət cəhətlərlə yanaşı bir sıra mənfi cəhətlər də mövcud ola bilər. Windows üzərində işlətdiyimiz bəzi məşhur proqramları, oyunları Linux üzərində işlədə bilməmə ehtimalımız var. Nəzərə alsaq ki, əksəriyyət Windows işlədir, proqramların çoxu Windows sistemləri üçündür, buna görə də, vaxtaşırı Linux üzərində də bu tip problemlər yaşaya bilərik. Ancaq yenə də, Linux üçün də bir sıra oyunlar, proqramlar yaradılır. Bəzən standart istifadəçi üçün Linux çətin ola bilər, ancaq, bir proqramçı üçün linux kifayət qədər yaxşı bir sistemdir. Qeyd edək ki,

Linux üzərindəki “Wine” proqramı vasitəsilə də bəzi Windows proqramlarını Linux üzərində işlədə bilərik.

İlk olaraq Ubuntu.com saytıdan Ubuntunu yükləyib komputerinizə quraşdırı bilərsiniz. Ubuntunu yükləyərkən komputerinizdən Windows sistemini silməyə ehtiyacımız olmayacaq. Yeni Linux sistemimizi Windows sisteminin yanına quraşdırı bilərik. Youtube üzərində Ubuntu sisteminin quraşdırılmasına dair bir çox videolar mövcuddur. Bu videolara baxaraq Ubuntu sistemini rahatlıqla komputerinizə quraşdırı bilərsiniz. İlk başda driver falan məsələsi olmayacağına görə də, quraşdırma çox rahat olacaq.

Əməliyyat sistemindən əlavə, əgər maraqlı gəlicə kernel.org saytıdan Linux nüvəsini yükləyib kodlarına baxı bilərik. Ancaq, kodların mürəkkəbliyini və böyüklüyünü nəzərə alaraq ürəyi zəif olan şəxslərə məsləhət görmürük :)

Ubuntu sistemini komputerə quraşdırdıqdan sonra sistemi ümumi nəzər keçirək bilərik. Sistemdə daxil olacağımız ən vacib alətlərdən biri Terminal olacaq. Sistemdə axtarış yerinə Terminal yazaraq və ya alt+ctrl+t düymələrinə bir yerdə basaraq Terminala daxil ola bilərik.



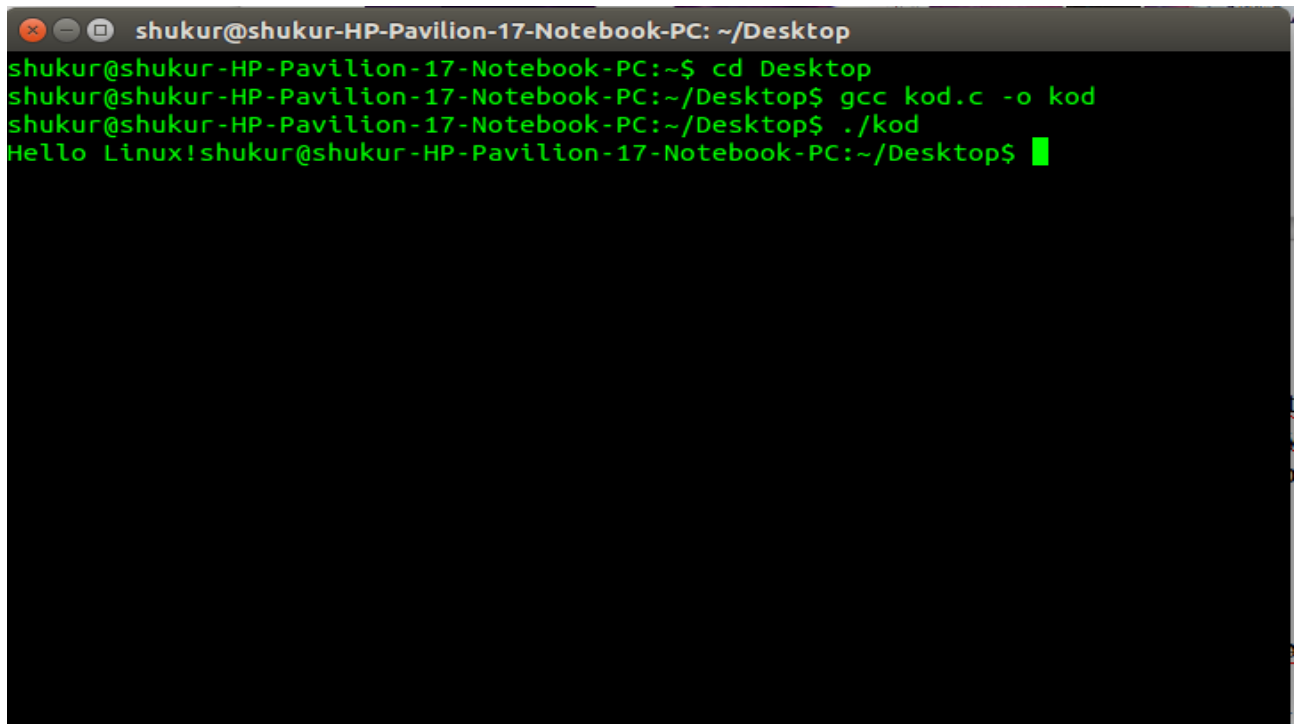
Yuxarıdakı pəncərə terminal pəncərəsidir. Bu pəncərə Windows sistemindəki cmd pəncərəsinə oxşardır. Pəncərə daxilində müxtəlif kodlar yazaraq müxtəlif əməliyyatlar yerinə yetirə bilərik. Məsələn, ls yazıb enter basdıqda, faylların siyahısı çıxacaq. Əlavə olaraq, cd Desktop yazdıqda masaüstünə gəlmiş olacağıq.

İndi isə C kodunun Linux üzərində kompilyasiya qaydasına baxaq. İlk olaraq Terminal üzərində aşağıdakı əmri yazaq və enter düyməsinə basaq:

```
sudo apt-get install gcc
```

Bundan sonra parolumuzu daxil edək. Parolumuzu daxil etdikdən sonra C kompilyatoru sistemimizə yüklənmiş olacaq.

Bundan sonra gedit editorunu açaq. Ubuntuda solda yuxarıda olan axtarış bölməsinə gedit yazaq və proqrama daxil olaq. Ondan sonra sadə bir C kodunu yazaq və Desktopda kod.c adı ilə yadda saxlayaq. Yadda saxladıqdan sonra isə Terminalda sıra ilə aşağıdakıları yerinə yetirək:

A terminal window titled 'shukur@shukur-HP-Pavilion-17-Notebook-PC: ~/Desktop'. The prompt is 'shukur@shukur-HP-Pavilion-17-Notebook-PC:~\$'. The user enters 'cd Desktop', changing the directory to '~/Desktop'. The prompt becomes 'shukur@shukur-HP-Pavilion-17-Notebook-PC:~/Desktop\$'. The user enters 'gcc kod.c -o kod', compiling the C file 'kod.c' into an executable named 'kod'. The prompt becomes 'shukur@shukur-HP-Pavilion-17-Notebook-PC:~/Desktop\$'. The user enters './kod', running the executable. The output is 'Hello Linux!shukur@shukur-HP-Pavilion-17-Notebook-PC:~/Desktop\$' followed by a green cursor.

```
shukur@shukur-HP-Pavilion-17-Notebook-PC: ~/Desktop
shukur@shukur-HP-Pavilion-17-Notebook-PC:~$ cd Desktop
shukur@shukur-HP-Pavilion-17-Notebook-PC:~/Desktop$ gcc kod.c -o kod
shukur@shukur-HP-Pavilion-17-Notebook-PC:~/Desktop$ ./kod
Hello Linux!shukur@shukur-HP-Pavilion-17-Notebook-PC:~/Desktop$
```

Şəkilə gördüyümüz kimi kod kompilyasiya olunaraq işə salınır və Hello Linux yazısı çap olunur. Beləliklə Linux üzərində ilk C kodumuzu kompilyasiya etmiş oluruq.

Qeyd: Gcc yalnız C kodlarını kompilyasiya edir. C++ kodları üçün isə G++ kompilyatoruna ehtiyacımız var. Bunu aşağıdakı şəkildə yükləyib istifadə edə bilərik:

```
sudo apt-get install g++
```

```
g++ kod.cpp -o kod2
```

```
./kod2
```

Gördüyümüz kimi, burada g++ kompilyatorunu yüklədikdən sonra, kod.cpp adında yadda saxladığımız c++ proqramı kompilyasiya edərək işlətməmiş oluruq.

OpenGL və oyun proqramlaşdırması

İstər C/C++ ilə olsun, istər başqa bir dillə, oyun proqramlaşdırması tamam ayrı bir dünyadır. Burada çox şey sizin fantaziyanızdan asılıdır. Fantaziyanız, xəyal gücünüz nə qədər çoxdursa o qədər yaxşıdır. Oyun sənayesinə baxdıqda insanın heyratə gəlməməyi mümkün deyil. Bugünkü oyunların möhtəşəmliyi qarşısında insan heyratə gəlir.

Hal-hazırda oyun proqramlaşdırmaq üçün Unity, Unreal Engine, CryEngine kimi bir çox oyun mühərriki yaradılmışdır. Oyun mühərriki ilə oyunu hazırlamaq daha asan olur. Hətta Game Maker adlı oyun mühərriki ilə illər öncə kod yazmadan ilk 2d oyunumu yaratdığımı xatırlayıram :)

Game Maker kiçik olsa da çox maraqlı, gözəl bir mühərrik idi. Mühərrikin içində özünə aid GML dili belə mövcuddur. Mühərrikin son versiyaları ilə hətta mobil telefonlara və Playstation oyun konsoluna belə oyun hazırlamaq mümkündür. Təsəvvür edin, oyunu yaradırsız və Playstationda oynayırsız, bunun zövqü tamam başqa olur.

Platformalar müxtəlifdir, ancaq bundan əlavə oyun məntiqini anlamaq mütləqdir. Yəni hərəkətlərin, hadisələrin necə baş verdiyini anlamaq lazımdır. Ondan sonra müxtəlif platformalarda oyunlar hazırlamaq mümkündür.

C++ istifadə edərək müxtəlif platformalarla, kitabxanalarla oyunlar yarada bilərik. Oyun mühərrikinə nümunə olaraq Unreal Engine misal göstərilə bilər. Unreal Engine mühərrikində proqramlaşdırma dili olaraq C++ istifadə olunmaqdadır. Ancaq, proqramlaşdırma dilindən əlavə mühərriklərin öz istifadə qaydalarını da öyrənmək lazımdır.

Kitabxanalara nümunə olaraq isə DirectX, Opengl və s. nümunə göstərmək olar. Məşhur oyunlardan biri olan Counter Strike oyunun bir versiyasında da OpenGL kitabxanasından istifadə olunduğunu demək yerinə düşər.

Biz də bu hissədə C dili ilə OpenGL kitabxanasından istifadə qaydalarına baxacağıq və kiçik bir oyun da hazırlayacağıq. OpenGL kitabxanasının ən yaxşı tərəfi multiplatform olmasıdır. Yəni, OpenGL kitabxanasını bütün əməliyyat sistemlərində istifadə edə bilərsiniz. Gəlin ilk olaraq C ilə linux üzərində OpenGL kitabxanasının quraşdırılmasına baxaq.

İlk olaraq Terminalı açaq (axtarışda terminal yazsa və ya ctrl+alt+T düymələrinə basarsınız) və sıra ilə aşağıdakı əməlləri yerinə yetirək.

```
sudo apt-get install freeglut3
```

```
sudo apt-get install freeglut3-dev
```

```
sudo apt-get install libglew1.5
```

```
sudo apt-get install libglew1.5-dev
```

```
sudo apt-get install libglu1-mesa
```

```
sudo apt-get install libglu1-mesa-dev
```

```
sudo apt-get install libgl1-mesa-glx
```

```
sudo apt-get install libgl1-mesa-dev
```

Sıra ilə bu əməlləri yazsaq və yerinə yetirək. Nəticədə OpenGL kitabxanası quraşdırılmış olacaq. Qeyd edək ki, Linuxdan əlavə Windows üzərində də OpenGL kitabxanasının quraşdırılması və istifadə olunması mümkündür.

Növbəti olaraq aşağıdakı kodu Text editorda yazsaq və open.c adı ilə Desktopda yadda saxlayaqa:

```
#include <GL/glut.h>
```

```
void display() {
```

```
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

```
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glMatrixMode(GL_MODELVIEW);
```

```
    glLoadIdentity();
```

```
    glFlush();
```

```
}
```

```
int main(int argc, char** argv) {
```

```

glutInit(&argc, argv);

glutInitWindowSize(900, 800);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

```

Faylı yadda saxladıqdan sonra yenidən Terminala daxil olaq. Bundan sonra `cd Desktop` əmri ilə masaüstünə gələk və aşağıdakı əmri yazaq:

```
gcc open.c -o open -L/usr/X11R6/lib/ -lGL -lGLU -lglut
```

Bundan sonra proqram kompilyasiya olunmuş olacaq və masaüstündə `open` adlı proqram yaranacaq. Həmin proqrama daxil olduqda qara bir pəncərə açılacaq. Bu bizim yuxarıda yazdığımız kodların nəticəsində yaranmış pəncərədir. Elə Terminal üzərindən `./open` yazaraq da proqramı açə bilərik.

Yuxarıda yazdığımız kod daxilindəki funksiyaların çox olmağı ilk başda adamı biraz qorxuda bilir :) Ancaq, buna qətiyyəən ehtiyac yoxdur, biraz keçdikdən insan bunlara öyrəşir. Eyni zamanda, bunları hər dəfə yazmağa ehtiyac yoxdur, bir dəfə yadda saxlayıb hər dəfə istifadə etmək olar. Mən elə edirəm :)

İndi isə gəlin yazdığımız kodların izahına keçək.

#include <GL/glut.h> - bu sətirlə OpenGL kitabxanasını kodumuza daxil edirik.

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); - Bu funksiya ilə fon rəngi təyin olunur, ilk üç parametr sıra ilə qırmızı, göy və yaşıl rəngləri təmsil edir. Bu parametrlər 0 və 1 arası qiymətləri alır və bu üç rəngin qarışığından fərqli rənglər alınır. Hər üç parametr 0 olduqda qara rəng ortaya çıxır. Buna görə də pəncərəmiz qara rəngdə idi.

glutInitWindowSize(900, 800); -Pəncərənin ölçüsü bu funksiya ilə təyin olunur.

glutCreateWindow("Game"); - Pəncərə yaradılır və pəncərənin başlığında yazı yazılır.

glutDisplayFunc(display); - Display funksiyası çağırılır. Diqqət edək ki, bir çox əməliyyatları display funksiyasının daxilində yazacağıq.

Digər parametrlər isə hələki qalsın, vacib hissələri öyrəndikdən sonra onları da nəzərdən keçirərik.

Fon rənginin dəyişdirilməsi

Yuxarıda da dediyimiz kimi, fon rəngi **glClearColor()** funksiyası ilə təyin olunur. Funksiyanın ilk 3 parametri sıra ilə qırmızı, yaşıl və göy dəyərlərini almaqdadır. Bu dəyərlər 0 və 1 arasında təyin olunur. Bu dəyərlərin müxtəlif qiymətləri ilə müxtəlif rəngləri almaq mümkündür. Aşağıda bir neçə müxtəlif rəngi göstərək.

Qırmızı rəng – `glClearColor(1.0f, 0.0f, 0.0f, 1.0f);`

Yaşıl rəng - `glClearColor(0.0f, 1.0f, 0.0f, 1.0f);`

Göy rəng - `glClearColor(0.0f, 0.0f, 1.0f, 1.0f);`

Ağ rəng - `glClearColor(1.0f, 1.0f, 1.0f, 1.0f);`

Qara rəng - `glClearColor(0.0f, 0.0f, 0.0f, 1.0f);`

Sarı rəng - `glClearColor(1.0f, 1.0f, 0.0f, 1.0f);`

Bənövşəyi rəng – `glClearColor(1.0f, 0.0f, 1.0f, 1.0f);`

Yuxarıda bir neçə əsas rəngi göstərmiş olduq. Funksiya daxilindəki ilk üç dəyəri 0 və 1 arasında dəyişərək fərqli rəng çalarlarını almağımız mümkündür. Aşağıda isə ümumi kodu göstərək. Yazdığımız kodun nəticəsində fon rəngi bənövşəyi olmuş olacaq.

```
#include <GL/glut.h>
```

```
void display() {
```

```
    glClearColor(1.0f, 0.0f, 1.0f, 1.0f);
```

```
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glMatrixMode(GL_MODELVIEW);
```

```
    glBegin(GL_POLYGON);
```

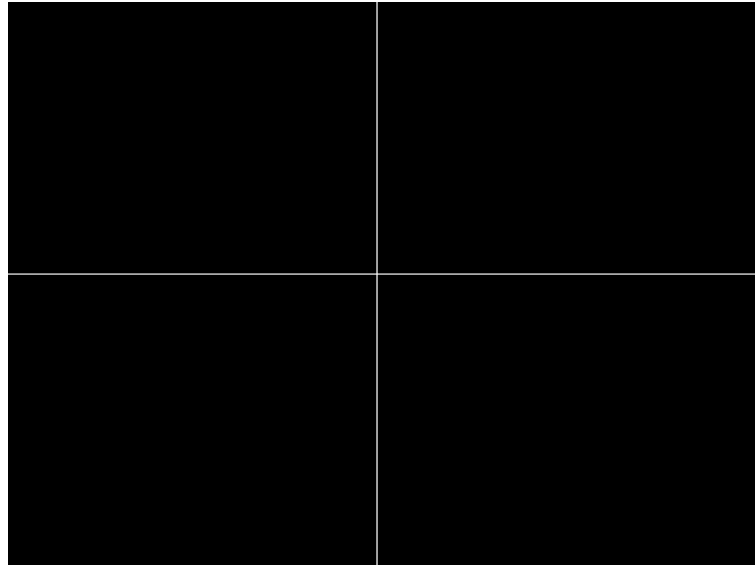
```
    glColor3f(1.0,0.0,1.0);
```

```
    glVertex2f(0.2,0.2);
```

```
glVertex2f(0.6,0.2);  
glVertex2f(0.4,0.8);  
glEnd();  
glLoadIdentity();  
glFlush();  
}  
  
int main(int argc, char** argv) {  
    glutInit(&argc, argv);  
    glutInitWindowSize(500, 500);  
    glutCreateWindow("Game");  
    glutDisplayFunc(display);  
    glutMainLoop();  
    return 0;  
}
```

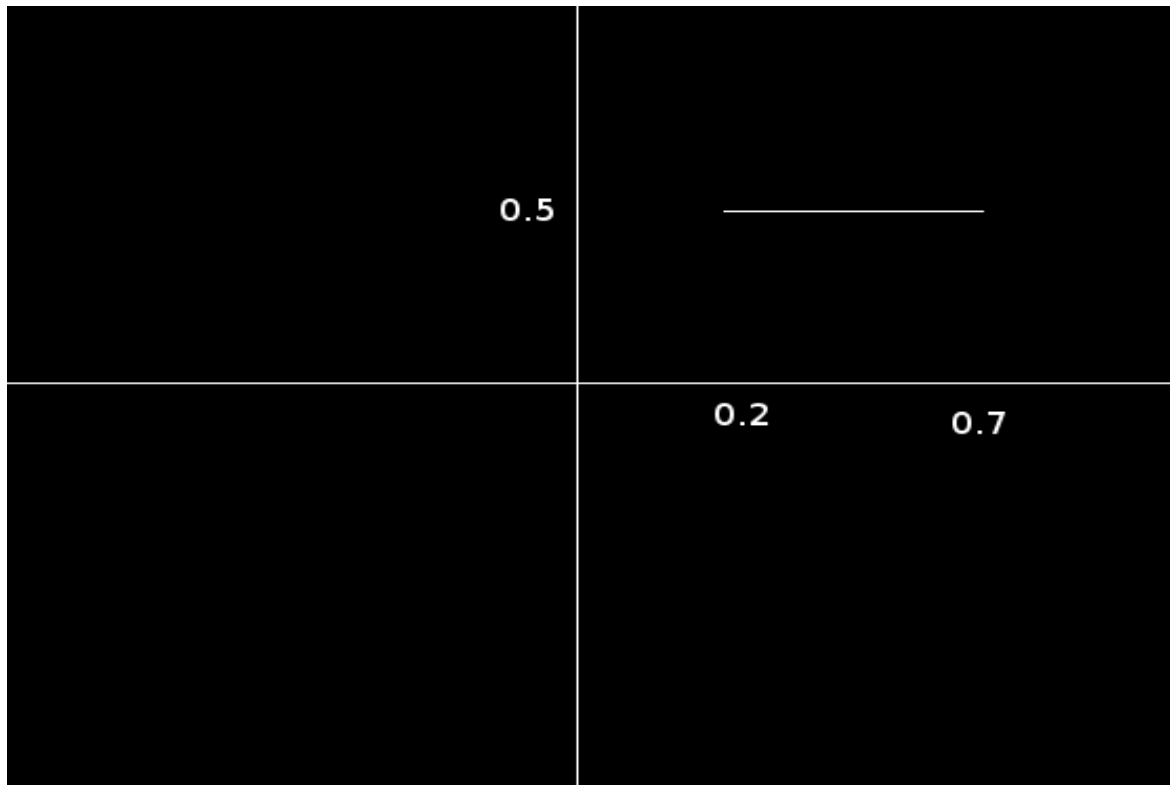
OpenGL ilə Fiqurların çəkilməsi

İlk olaraq OpenGL ilə fiqurların çəkilməsinə baxacağıq. İrəlidəki mövzularda bu fiqurlar vasitəsilə biz oyunlar belə hazırlayacağıq. Ancaq, kodun yazmaqdan öncə koordinat sisteminə baxaq. Program daxilindəki fiqurları koordinat sistemimizə əsasən çəkəcəyik.



Burada nöqtələr -1.0 və 1.0 aralığında dəyişməkdədir. Tam mərkəz nöqtənin koordinatları $(0.0, 0.0)$ qiymətlərini almaqdadır. Soldan sağa çəkilmiş horizontal xəttin başlanğıcı $(-1.0, 0.0)$ nöqtələrinə düşməkdədir. Burada birinci dəyər x , ikinci dəyər isə y -dir. Həmin xəttin sağ tərəfdə sonu isə $(1.0, 0.0)$ dəyərlərini almaqdadır. Aşağıdan yuxarıya doğru olan vertical xətdə isə, aşağı nöqtə $(0.0, -1.0)$ qiymətlərini, yuxarı nöqtə isə $(0.0, 1.0)$ qiymətlərini almaqdadır. Çəkəcəyimiz fiqurlar bu koordinat sistemindəki nöqtələrə əsasən çəkiləcək. Ona görə də, kodu yazmadan öncə ilk olaraq çəkəcəyimiz fiqurun nöqtələrini təyin etməliyik.

Düz xəttin çəkilməsi



İlk olaraq çəkəcəyimiz xətti koordinat sistemində çəkək və onun koordinatlarını təyin edək. Ondan sonra isə kodu yazaq.

Təxmini olaraq düz xətti çəkdikdən sonra təyin edə bilərik ki, xəttin başlanğıc koordinatları (0.2, 0.5), son koordinatları isə (0.7, 0.5) olmuş olur. Bu koordinatlara əsasən xəttimizi çəkə bilərik. Bunun display funksiyaımızın daxilinə aşağıdakı kodu yazmalıyıq:

```
glBegin(GL_LINES);  
glVertex2f(0.2,0.5);  
glVertex2f(0.7,0.5);  
glEnd();
```

Kodu izah edək. OpenGL daxilində fiqurlar glBegin() və glEnd() funksiyaları arasında yazılır. Yuxarıda glBegin funksiyaımız **GL_LINES** parametrini almaqdadır.

Bu parametr, fiqurumuzun xətt olduğunu göstərir. Növbəti olaraq **glVertex2f(0.2,0.5)** funksiyasından istifadə edirik. Funksiyaya parametr olaraq xəttin ilk nöqtəsinin koordinatlarını yazırıq. Növbəti olaraq isə eyni funksiyadan istifadə edirik, ancaq, bu dəfə parametr olaraq ikinci nöqtənin koordinatlarını yazırıq. İndi isə, ümumi kodumuza baxaq:

```
#include <GL/glut.h>

void display() {

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);

    glBegin(GL_LINES);

    glVertex2f(0.2,0.5);

    glVertex2f(0.7,0.5);

    glEnd();

    glLoadIdentity();

    glFlush();

}

int main(int argc, char** argv) {

    glutInit(&argc, argv);

    glutInitWindowSize(900, 800);

    glutCreateWindow("Game");

    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

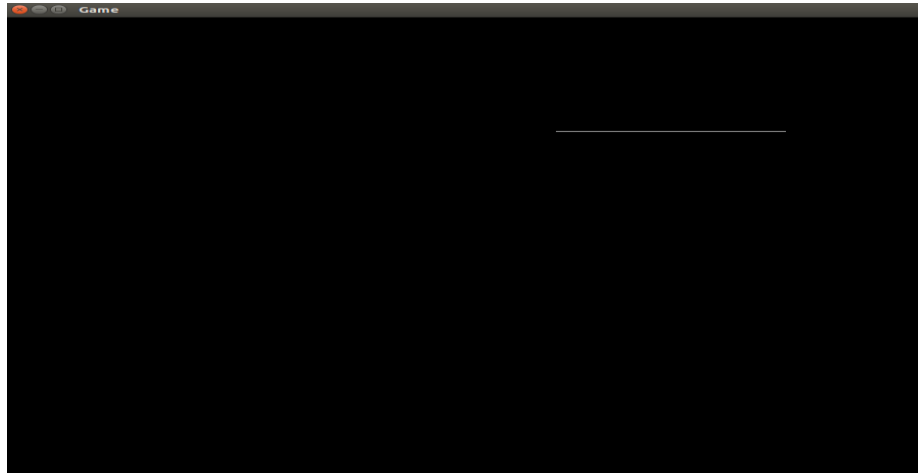
}
```

Kodu yazıb yadda saxladıqdan sonra kompilyasiya edək və proqramı açaq:


```
gcc open.c -o open -L/usr/X11R6/lib/ -lGL -lGLU -lglut
```

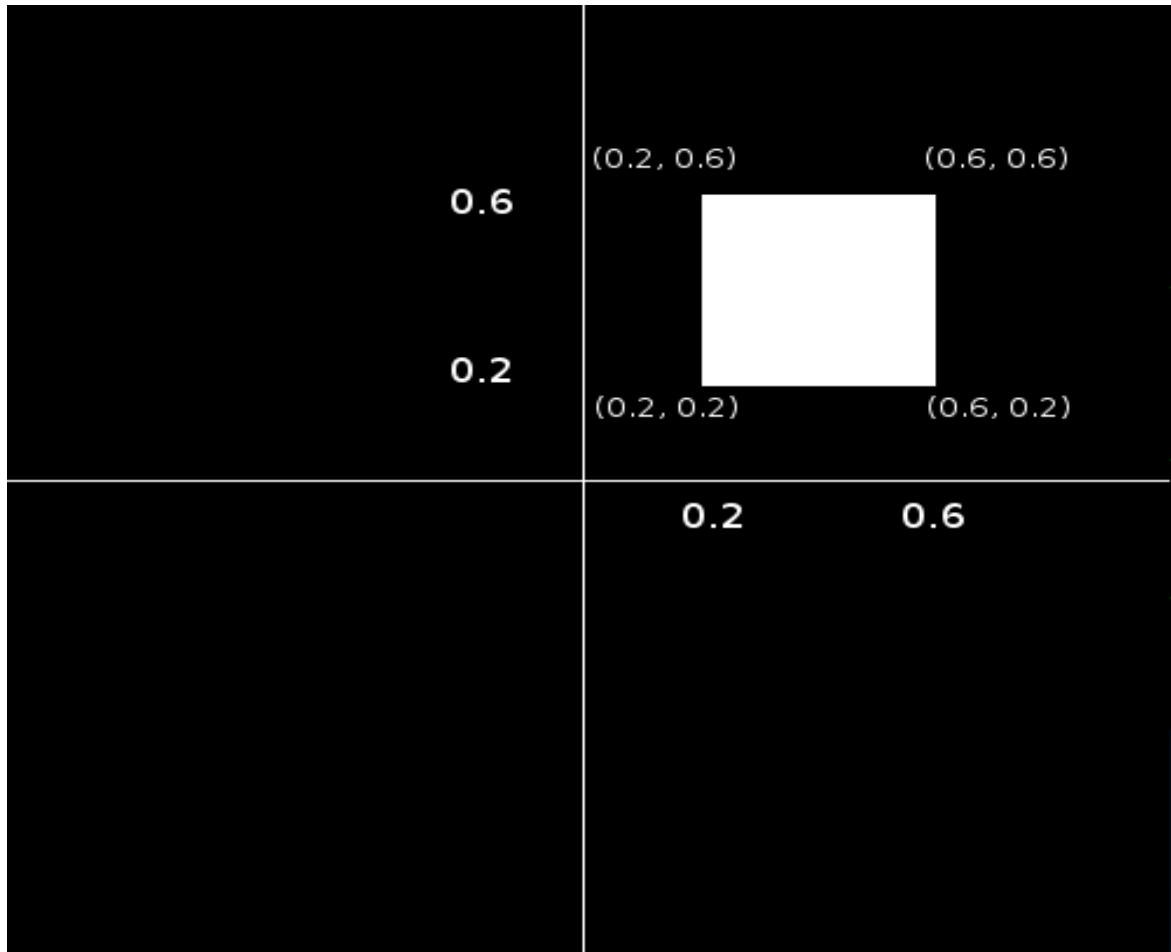
```
./open
```

Bundan sonra, içərisində xətt çəkilmiş proqramımız açılacaq.



Düzbucaqlının çəkilməsi

İndi isə, OpenGL ilə düzbucaqlının çəkilməsinə baxaq. İlk ökcə düzbucaqlını koordinat sistemində təxmini olaraq çəkək və onun koordinatlarını təyin edək.



Çəkəcəyimiz düzbucaqlının koordinatlarını təyin etdikdən sonra kodumuzu yazaq.

```
glBegin(GL_POLYGON);
```

```
glVertex2f(0.2,0.6);
```

```
glVertex2f(0.6,0.6);
```

```
glVertex2f(0.6,0.2);
```

```
glVertex2f(0.2,0.2);
```

```
glEnd();
```

Burada **glBegin()** funksiyasına parametr olaraq **GL_POLYGON** dəyərini ötürürük. Bu dəyər, çəkəcəyimiz fiqurun bir neçə nöqtədən ibarət olacağını bildirir. Fiqurumuz dörd nöqtədən ibarətdir. Buna görə də, **glVertex2f()** funksiyasından 4 dəfə istifadə edirik və 4 nöqtəni yazırıq. Beləliklə, düzbucaqlı fiqurumuz çəkilmiş olur. Ümumi kodumuz isə aşağıdakı şəkildə olur:

```
#include <GL/glut.h>

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glBegin(GL_POLYGON);

glVertex2f(0.2,0.6);

glVertex2f(0.6,0.6);

glVertex2f(0.6,0.2);

glVertex2f(0.2,0.2);

glEnd();

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

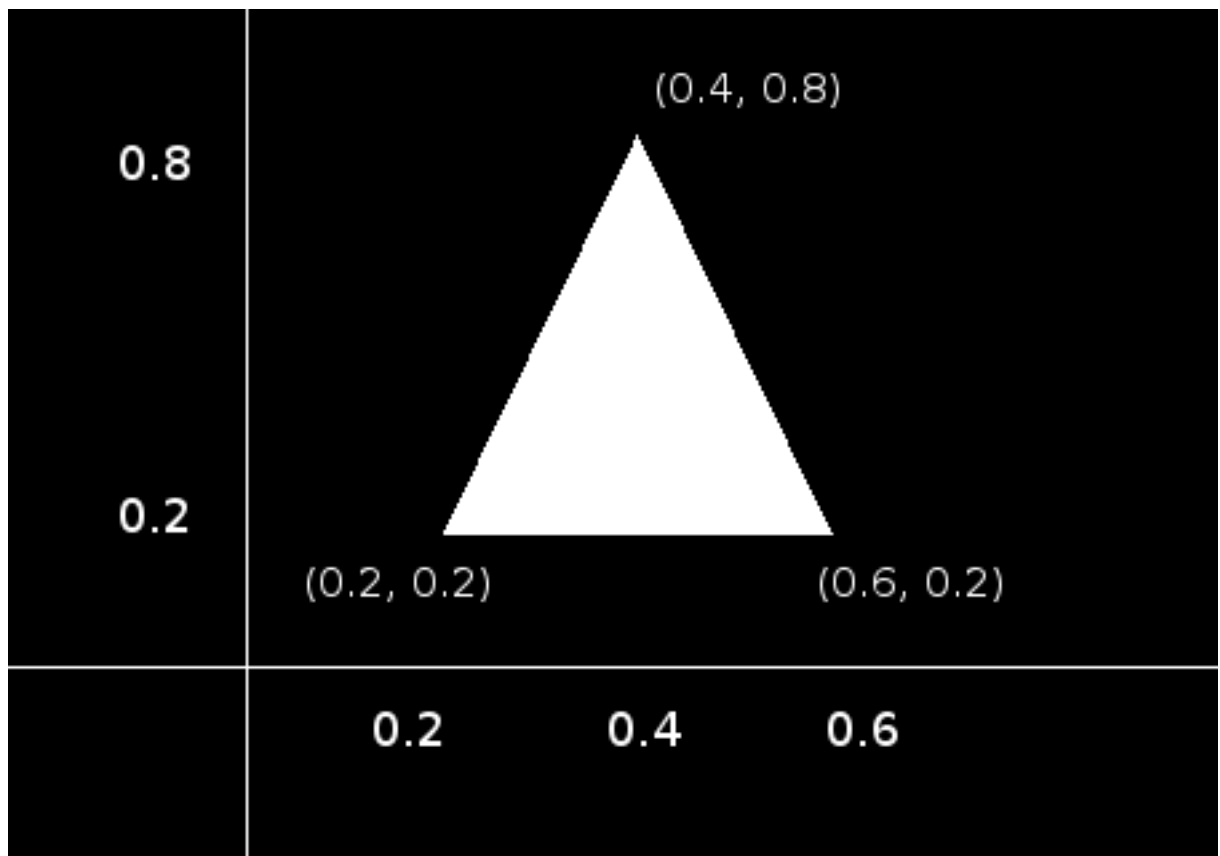
glutCreateWindow("Game");

glutDisplayFunc(display);

glutMainLoop();
```

```
return 0;  
}
```

Üçbucağın çəkilməsi



Üçbucağı çəkmək üçün ilk öncə onun koordinatlarını təyin edək.

Üçbucağın ortadakı nöqtəsinin kənardakı nöqtələrin ədədi ortası olduğuna diqqət edək.

$$\text{Ortadakı nöqtə} = (\text{kənar1} + \text{kənar2}) / 2 = (0.2 + 0.6) / 2 = 0.8 / 2 = 0.4$$

İndi isə kodu yazaq.

```
glBegin(GL_POLYGON);
```

```
glVertex2f(0.2,0.2);
```

```
glVertex2f(0.6,0.2);
```

```
glVertex2f(0.4,0.8);
```

```
glEnd();
```

Üçbucaq da çoxbucaqlı sayıldığı üçün **glBegin()** funksiyası burada da **GL_POLYGON** parametrini almaqdadır. **glVertex2f()** funksiyaları vasitəsilə isə, üçbucağın nöqtələri qeyd edilir. Ümumi kodumuz isə aşağıdakı kimi olur.

```
#include <GL/glut.h>

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glBegin(GL_POLYGON);

glVertex2f(0.2,0.2);

glVertex2f(0.6,0.2);

glVertex2f(0.4,0.8);

glEnd();

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");

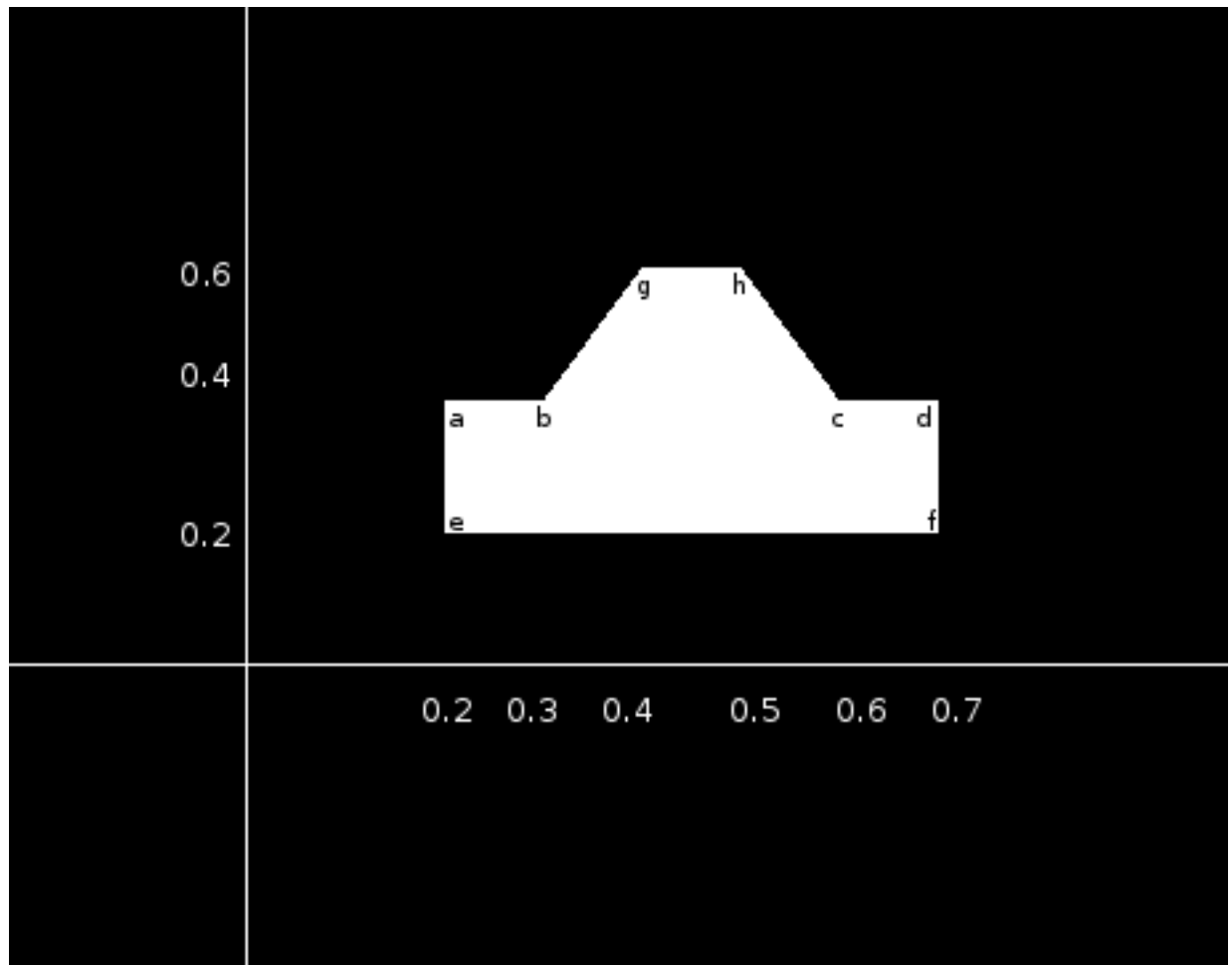
glutDisplayFunc(display);

glutMainLoop();
```

```
return 0;  
}
```

Daha mürəkkəb çoxbucaqlı fiqurlar

İndi isə, bir neçə tərəfli daha mürəkkəb bir fiqur çəkək. Kodu yazmadan öncə, fiquru koordinat sistemində çəkək və koordinatlarını müəyyən edək.



Fiquru koordinat sistemində çəkdikdən sonra nöqtələrə görə koordinatları yazaq.

$E=(0.2, 0.2)$; $a=(0.2, 0.4)$; $b=(0.3, 0.4)$; $g=(0.4, 0.6)$; $h=(0.5, 0.6)$; $c=(0.6, 0.4)$; $d=(0.7, 0.4)$; $f=(0.7, 0.2)$;

Koordinat sistemindəki nöqtələri müəyyən etdikdən sonra, aşağıdakı kod vasitəsilə şəkildəki fiqurumuzu çəkirik.

```
#include <GL/glut.h>
```

```
void display() {
```

```
glClearColor(0.0f, 0.0f, 0.0f, 0.0);
```

```
glClear(GL_COLOR_BUFFER_BIT);
```



```

glMatrixMode(GL_MODELVIEW);

glBegin(GL_POLYGON);

glVertex2f(0.2,0.2);

glVertex2f(0.2,0.4);

glVertex2f(0.3,0.4);

glVertex2f(0.4,0.6);

glVertex2f(0.5,0.6);

glVertex2f(0.6,0.4);

glVertex2f(0.7,0.4);

glVertex2f(0.7,0.2);

glEnd();

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

```

Dairənin çəkilməsi

OpenGL vasitəsilə birbaşa dairə çəkmək üçün funksiya mövcud deyil. Buna görə də dairəni düz xəttlər vasitəsilə çəkməliyik. Ancaq, dairəni düz xəttlər vasitəsilə çəkərkən bəzi triqonometriya qanunlarından istifadə etməliyik. Aşağıdakı kodda radiusu 0.3, x və y dəyərləri 0.1 olan dairə çəkilir.

```
#include <GL/glut.h>

#include <math.h>

void drawCircle(float r, float x, float y) {

    float i = 0.0f;

    glBegin(GL_TRIANGLE_FAN);

    glVertex2f(x, y); // Center

    for(i = 0.0f; i <= 360; i++)

        glVertex2f(r*cos(M_PI/180.0 * i) + x, r*sin(M_PI/180.0 * i) + y);

    glEnd();

}

void display() {

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);

    glLoadIdentity();

    drawCircle(0.3,0.1,0.1);

    glFlush();

}

int main(int argc, char** argv) {

    glutInit(&argc, argv);
```

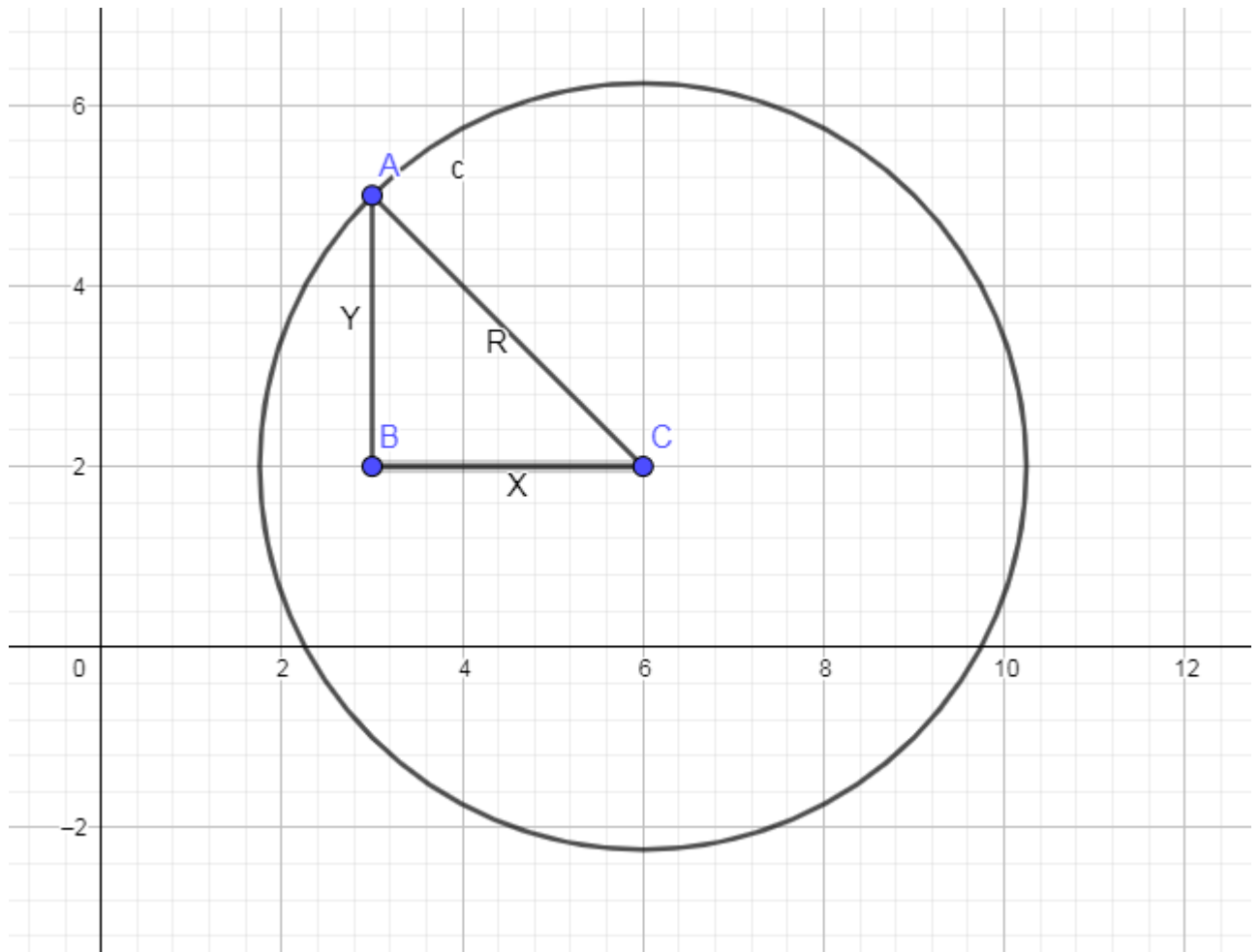
```
glutInitWindowSize(900, 800);  
glutCreateWindow("Game");  
glutDisplayFunc(display);  
glutMainLoop();  
return 0;  
}
```

Burada riyazi ifadələrdən də istifadə olunduğu üçün kompilyasiya zamanı `lm` kitabxanasından da istifadə etməliyik. Kompilyasiya kodumuz aşağıdakı şəkildə olacaq:

```
gcc open.c -o open -L/usr/X11R6/lib/ -lGL -lGLU -lglut -lm
```

Yazdığımız **drawCircle()** funksiyasını `display` funksiyası daxilində işlədərək asanlıqla dairəmizi çəkə bilirik. Ancaq, **drawCircle()** funksiyasını anlamaq üçün biraz triqonometriya biliklərinə ehtiyacımız olacaq.

Dairə çəkmək üçün ilk olaraq mərkəzi bir nöqtə götürürük. Həmin nöqtənin ətrafında müəyyən bir R radiusu ilə 360 dərəcə boyunca nöqtələr düzəndə nöqtələrin birləşməsindən dairə fiqurunu almış oluruq. Aşağıdakı şəklə baxaq.



Burada üçbucağın hipotenuzu R olaraq işarə olunmuşdur. Üçbucağın hipotenuzu çəkdiyimiz dairənin radiusu olmuş olur. Burada C nöqtəsinə bitişik bucağı, c bucağı olaraq işarə edək. O zaman, aşağıdakı bərabərlikləri yazı bilərik:

$$\cos(c) = x/R, x = \cos(c) * R$$

$$\sin(c) = y/R, y = \sin(c) * R$$

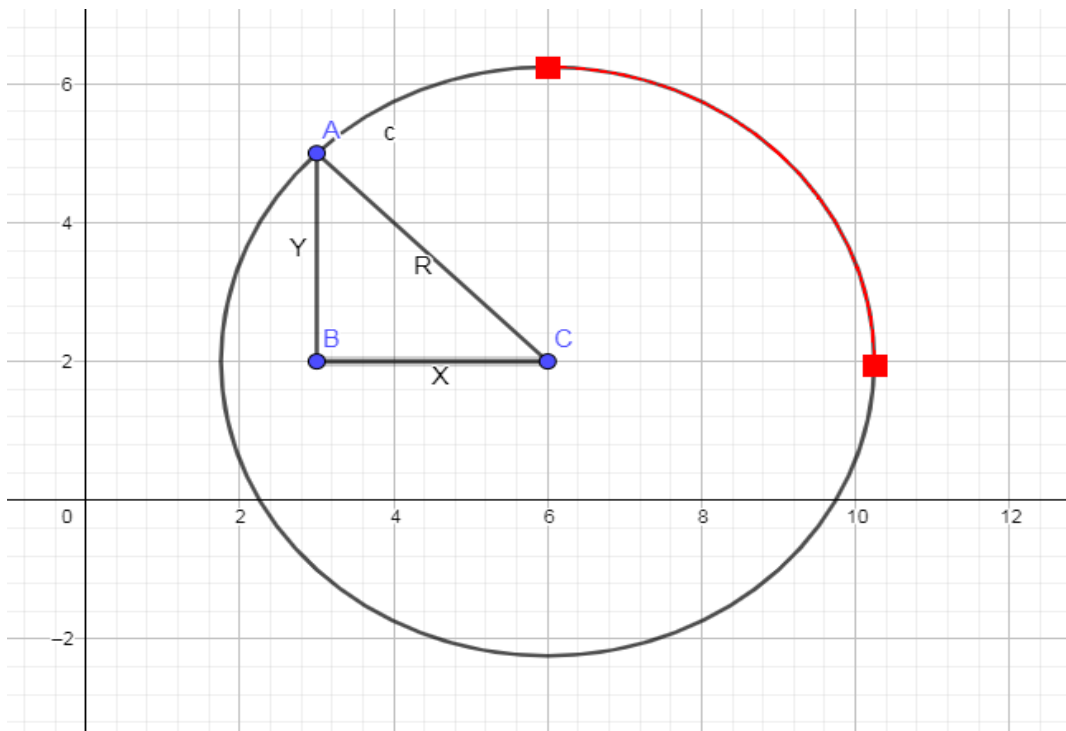
Artıq, triqonometrik ifadələr vasitəsilə dairənin nöqtələrini müəyyən edə bilərik. Kodda yazdığımız ifadəyə bir daha diqqət yetirək:

```
glVertex2f(r*cos(M_PI/180.0 * i) + x, r*sin(M_PI/180.0 * i) + y);
```

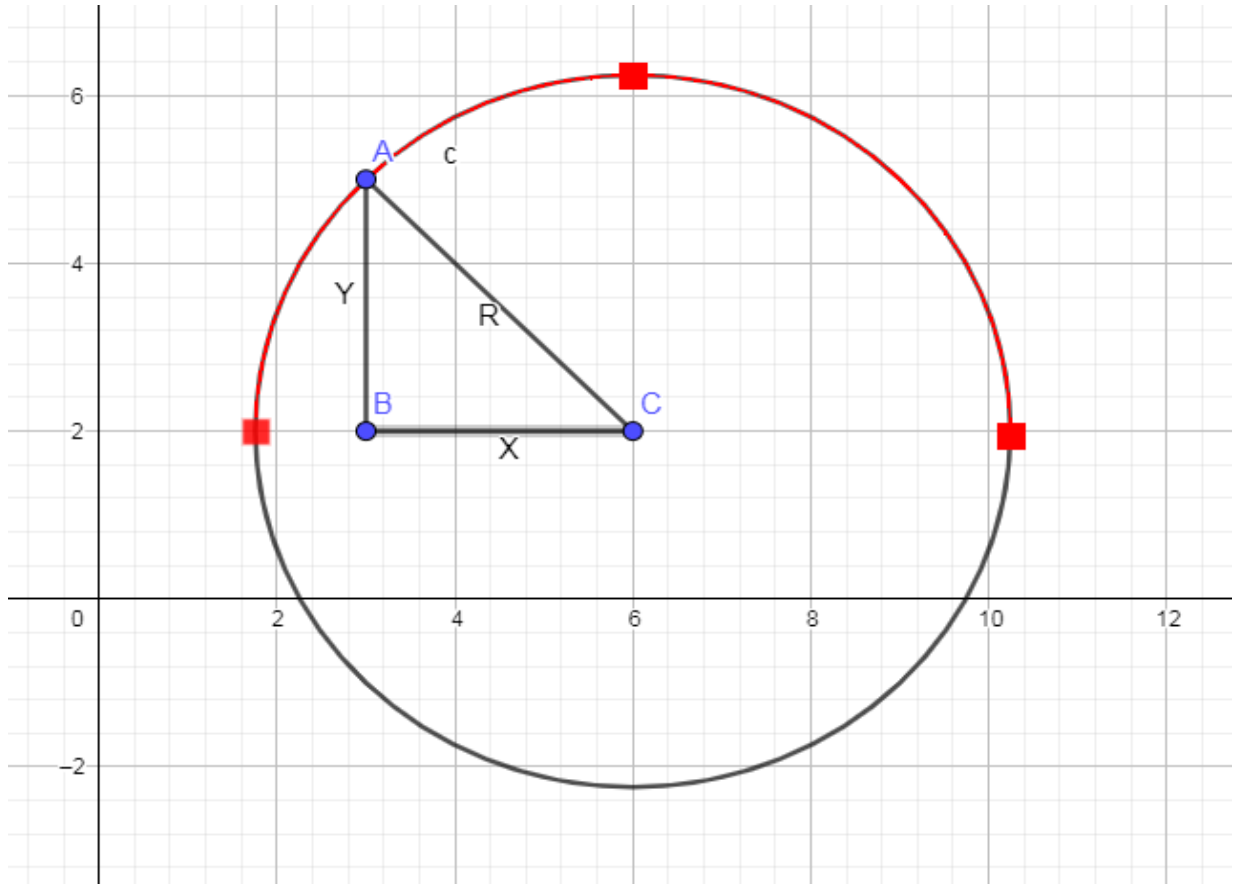
Burada I dəyəri 0-dan 360-a qədər dövr etdirilərək dairə çəkilmiş olur. Yazdığımız x və y çevrənin mərkəzinin koordinatlarıdır. $R=0.3$, $X=0.1$, $Y=0.1$ qəbul edək. Bu o deməkdir ki, mərkəzi nöqtələr $X=0.1$ və $Y=0.1$ olacaq. Radius isə 0.3 olmuş olacaq. C dilində kosinus funksiyası dəyəri radianla almaqdadır. Biz isə dəyəri dərəcə ilə verəcəyimizə görə, I dəyişənin dəyərini dərəcəyə çevirmək üçün, bu dəyəri $Pi/180$ dəyərinə vururuq. Burada **M_PI** ifadəsi pi ədədinin dəyərini göstərir.

İlk dövrdə I dəyişənin dəyəri 0 olur. Buna görə də $\cos(0)$ alınır. $\cos(0)$ isə 1-ə bərabərdir. Sonra isə radius bu dəyərə vurulur və 0.3 alınır. Bu dəyərin üzərinə isə mərkəzi x nöqtəsi gəlinir. Burada x nöqtəsi 0.1 -ə bərabərdir və nəticədə 0.4 alınır. İkinci hissədə isə $r \cdot \sin(M_PI/180.0 * i) + y$ ifadəsinin dəyəri 0.1 olur. Nəticədə mərkəzi nöqtə olan $(0.1, 0.1)$ nöqtəsi ilə $(0.4, 0.1)$ nöqtələri bir düz xəttlə birləşdirilir.

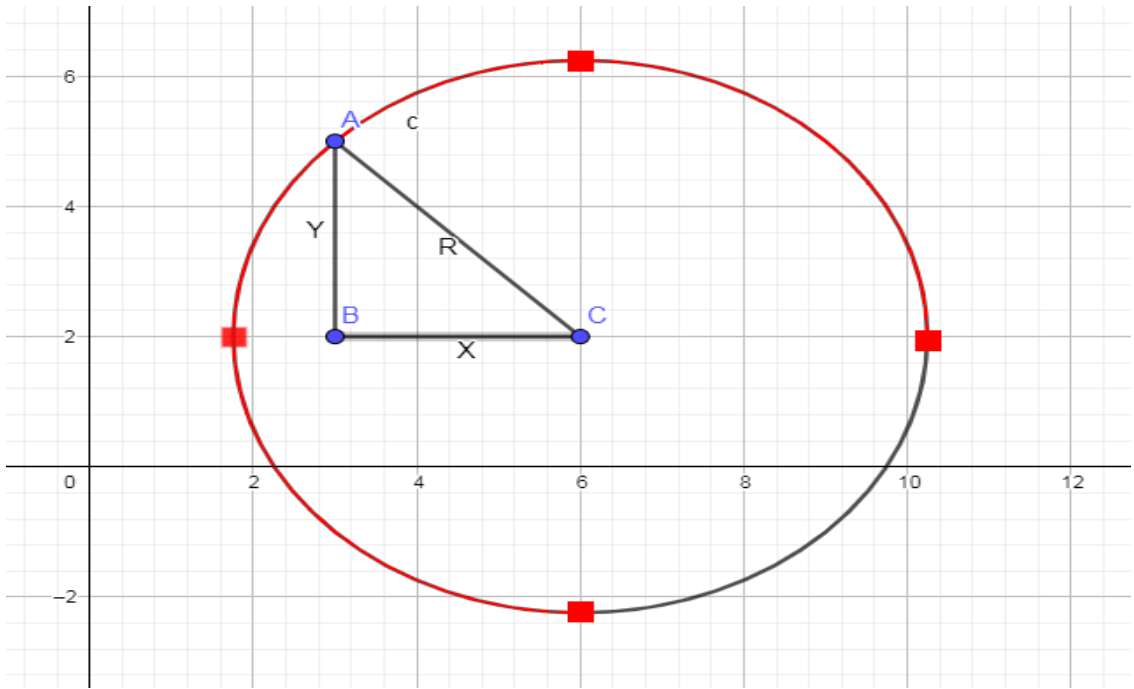
Birinci dövrə bitdikdən sonra ikinci dövrə başlayır. İkinci dövrdə I dəyişənin qiyməti 1 olaraq təyin olunur. Hesablamalardan sonra mərkəz nöqtə ilə alınmış yeni nöqtə birləşdirilir. Alınmış yeni nöqtənin x dəyəri əvvəlkindən biraz az, y dəyəri isə əvvəlkindən az miqdarda çox olmuş olur. Üçüncü dövrdə də eyni əməliyyatlar yerinə yetirilir. 90 dövrə boyunca dəyişmələr bu şəkildə dəyişir və şəkildə göstərildiyi kimi, qırmızı ilə göstərilmiş hissə artıq çəkilmiş olur.



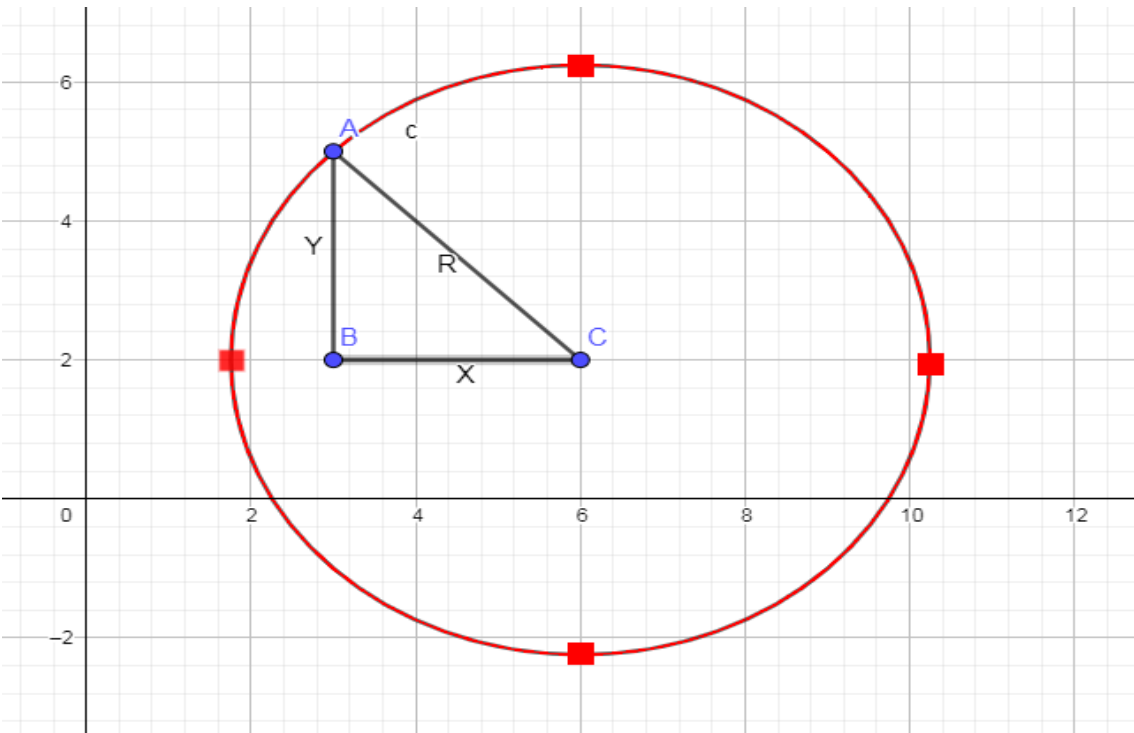
Növbəti 90 dövrədə isə eyni hesablamalar baş verir. Ancaq bu hissədə, x dəyəri ilə bərabər y dəyəri də azalır və sol tərəf də çəkilmiş olur.



180 dövrə artıq tamamlanmış olur. Növbəti 180 və 270-ci dövrlər arasında da eyni hesablamalar aparılır. Ancaq bu dəfə, həm x dəyəri, həm də y dəyəri 90 dövrə boyunca azalır və üçüncü hissə də çəkilmiş olur.



Sonuncu 270 və 360 arası 90 dövrədə isə x və y dəyərləri hər ikisi artır və sonda dairə tamamlanmış olur.



Çəkilən fiqurların rənglənməsi

Çəkdiyimiz fiqurları müxtəlif rənglərdə yaratmağımız mümkündür. Bunun üçün **glColor3f()** funksiyasından istifadə edə bilərik. Funksiya üç parametralıdır. Birinci parametrlər red (qırmızı dəyəri), ikinci parametrlər green (yaşıl dəyəri), üçüncü parametrlər isə blue (göy dəyəri) olaraq təyin olunur. Bu dəyərlər 0 və 1 arasında qiymətlər alırlar. Funksiyanı aşağıdakı şəkildə istifadə edərək çəkəcəyimiz üçbucağı yaşıl rəngdə edə bilərik.

```
glBegin(GL_POLYGON);
```

```
glColor3f(0.0,1.0,0.0);
```

```
glVertex2f(0.2,0.2);
```

```
glVertex2f(0.6,0.2);
```

```
glVertex2f(0.4,0.8);
```

```
glEnd();
```

Ümumi kodumuz isə aşağıdakı şəkildə olacaq:

```
#include <GL/glut.h>
```

```
void display() {
```

```
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

```
glClear(GL_COLOR_BUFFER_BIT);
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glBegin(GL_POLYGON);
```

```
glColor3f(0.0,1.0,0.0);
```

```
glVertex2f(0.2,0.2);
```

```
glVertex2f(0.6,0.2);
```

```
glVertex2f(0.4,0.8);
```



```

glEnd();

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

```

Nəticədə yaşıl rəngdə üçbucaq çəkmiş olacağıq. Aşağıda bir neçə rəngi göstərək.

Qırmızı rəng – glColor3f(1.0,0.0,0.0);

Yaşıl rəng - glColor3f(0.0,1.0,0.0);

Göy rəng - glColor3f(0.0,0.0,1.0);

Ağ rəng - glColor3f(1.0,1.0,1.0);

Qara rəng - glColor3f(0.0,0.0,0.0);

Sarı rəng - glColor3f(1.0,1.0,0.0);

Bənövşəyi rəng – glColor3f(1.0,0.0,1.0);

Rəng dəyərlərini 0 və 1 arasında azaldaraq və ya çoxaldaraq müxtəlif rəng çalarları əldə edə bilərsiniz.

Pəncərədə yazıların yazılması

Yaratdığımız pəncərədə hər-hansı bir nöqtədə yazılar yazmağımız mümkündür. Bunun üçün **glutBitmapString()** funksiyasından istifadə edə bilərik. Aşağıdakı koda baxaq.

```
#include <GL/glut.h>

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glColor3f(0,255,0);

glRasterPos2f(0.0, 0.0);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, "Opengl");

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutMainLoop();

return 0;

}
```

Yazdığımız kodu yadda saxlayıb proqramı kompilyasiya edək. Nəticədə, (0.0, 0.0) nöqtəsində “Opengl” yazısı yazılacaq. Yazacağımız yazının koordinatlarını **glRasterPos2f(0.0, 0.0);** funksiyası vasitəsilə təyin edirik. Ondan sonra isə **glutBitmapString();** funksiyası vasitəsilə yazını yazırıq. Funksiyadakı ikinci parametr yazıdır. Birinci parametr isə, yazının fontunu və ölçüsünü göstərir. Yazıdan əvvəl **glColor3f(0,255,0);** yazdığımıza görə yazı yaşıl rəngdə olur. Aşağıda bəzi yazı fontları göstərilmişdir:

GLUT_BITMAP_8_BY_13

GLUT_BITMAP_9_BY_15

GLUT_BITMAP_TIMES_ROMAN_10

GLUT_BITMAP_TIMES_ROMAN_24

GLUT_BITMAP_HELVETICA_10

GLUT_BITMAP_HELVETICA_12

GLUT_BITMAP_HELVETICA_18

Bu fontlardan istifadə edərək yazıları müxtəlif ölçülərdə, müxtəlif fontlarda yazmağımız mümkündür.

Klaviatura əməliyyatları

Qrafiki interfeysdə hər bir şey hadisələr əsasında baş verməkdədir. Hadisələrə misal olaraq proqramın başlaması, proqramın bağlanması, klaviaturada hər-hansı bir düyməyə klik olunması, mausun hərəkəti, zamanın keçməsi və s. göstərilə bilər. Bu mövzuda isə, OpenGL ilə klaviatura əməliyyatlarına baxacağıq. Sadə klaviatura funksiyası üçün aşağıdakı kodu yazmaq və kompilyasiya etmək.

```
#include <GL/glut.h>

void keyboard(unsigned char key, int x, int y){
    exit(0);
}

void display() {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Game");
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

```
}
```

Proqramı kompilyasiya edib işə salaq. Proqramı açdıqdan sonra hər-hansı bir düyməyə basdıqda proqram bağlanacaq. İndi isə əsas hissələri nəzərdən keçirək. İlk baxacağımız klaviatura üçün yazdığımız funksiyadır.

```
void keyboard(unsigned char key, int x, int y){  
  
    exit(0);  
  
}
```

Burada keyboard adlı funksiyamızı yazırıq və funksiya üç parametr almaqdadır. Birinci key adlı parametrimiz basılan düyməni bildirir. Növbəti x və y parametrləri isə mous oxunun klaviatura düyməsinə basılarkən olduğu nöqtənin koordinatlarıdır. Funksiya daxilində sadəcə bir əməliyyat yazmışıq. Bu **exit(0);** əməliyyatı vasitəsilə proqram bağlanmaqdadır. Proqramımızın main hissəsində isə `glutKeyboardFunc(keyboard);` funksiyası vasitəsilə yazdığımız funksiyanı proqramımıza əlavə edirik. Bu funksiya parametr olaraq yuxarıda yazdığımız keyboard funksiyasının adını qeyd edirik. Beləliklə ilk klaviatura əməliyyatı olan OpenGL proqramımızı yazmış oluruq.

İndi isə elə edək ki, sadəcə Esc düyməsinə basıldıqda proqram bağlansın, digər düymələr basıldıqda isə heç nə olmasın. Diqqətə çatdıraq ki, klaviatura düymələrinə müraciət edərkən birbaşa adı ilə yox, ascii kodu ilə müraciət edirik. Yuxarıda yazdığımız kodun sadəcə keyboard funksiyasında aşağıdakı dəyişikliyi edək, digər yerləri olduğu kimi saxlayaq.

```
void keyboard(unsigned char key, int x, int y){  
  
    if(key==27){  
  
        exit(0);  
  
    }  
  
}
```

Kodu kompilyasiya edib proqramımızı işlədək. Proqramı açdıqdan sonra yalnız Esc düyməsinə basdıqda proqram bağlanacaq, digər düymələrə basdıqda isə heçnə baş verməyəcək. Bunun səbəbi, proqramı bağlamaq üçün olan `exit(0)` funksiyasını **`if(key==27)`** şərti altında yazmağımızdır. 27 ədədi Esc düyməsinin kodudur. Biz bu şərtə basılacaq düymənin kodunun 27-ə bərabər olması halında, yəni, düymənin Esc olması halında proqramı bağlayırıq. Əgər şərt ödənməzsə, o zaman heç bir şey baş verməyəcək.

Ascii cədvəli aşağıda göstərilmişdir.

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
0 0 NUL	16 10 DLE	32 20 (space)	48 30 0
1 1 SOH	17 11 DC1	33 21 !	49 31 1
2 2 STX	18 12 DC2	34 22 "	50 32 2
3 3 ETX	19 13 DC3	35 23 #	51 33 3
4 4 EOT	20 14 DC4	36 24 \$	52 34 4
5 5 ENQ	21 15 NAK	37 25 %	53 35 5
6 6 ACK	22 16 SYN	38 26 &	54 36 6
7 7 BEL	23 17 ETB	39 27 '	55 37 7
8 8 BS	24 18 CAN	40 28 (56 38 8
9 9 TAB	25 19 EM	41 29)	57 39 9
10 A LF	26 1A SUB	42 2A *	58 3A :
11 B VT	27 1B ESC	43 2B +	59 3B ;
12 C FF	28 1C FS	44 2C ,	60 3C <
13 D CR	29 1D GS	45 2D -	61 3D =
14 E SO	30 1E RS	46 2E .	62 3E >
15 F SI	31 1F US	47 2F /	63 3F ?
ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
64 40 @	80 50 P	96 60 `	112 70 p
65 41 A	81 51 Q	97 61 a	113 71 q
66 42 B	82 52 R	98 62 b	114 72 r
67 43 C	83 53 S	99 63 c	115 73 s
68 44 D	84 54 T	100 64 d	116 74 t
69 45 E	85 55 U	101 65 e	117 75 u
70 46 F	86 56 V	102 66 f	118 76 v
71 47 G	87 57 W	103 67 g	119 77 w
72 48 H	88 58 X	104 68 h	120 78 x
73 49 I	89 59 Y	105 69 i	121 79 y
74 4A J	90 5A Z	106 6A j	122 7A z
75 4B K	91 5B [107 6B k	123 7B {
76 4C L	92 5C \	108 6C l	124 7C
77 4D M	93 5D]	109 6D m	125 7D }
78 4E N	94 5E ^	110 6E n	126 7E ~
79 4F O	95 5F _	111 6F o	127 7F □

Cədvəl boyunca birinci sütunda işlədəcyimiz ədəd, üçüncü sütunda isə düymənin adı yazılmışdır. Məsələn, ilk sütunda 27 yazılmış sətirə baxsaq, qarşısında, üçüncü stunda ESC düyməsinin yazıldığını görəcəyik. Bu kodlardan istifadə edərək, müxtəlif düymələrlə müxtəlif əməliyyatlar apara bilərik.

Yuxarıda yazdığımız kodda aşağıdakı kimi bir dəyişiklik edək.

```
void keyboard(unsigned char key, int x, int y){
```

```

if(x>250){
    if(key==27){
        exit(0);
    }
}
}

```

Proqramı kompilyasiya edib işlədək. Bu dəfə mausumuzun oxunu proqramın sol tərəfində saxlayaq və Esc düyməsinə basaq. Esc düyməsinə basdıqda proqramdan çıxılmadığını görəcəyik. İndi isə mausumuzun oxunu proqramın sağ tərəfinə gətirək. Bu dəfə Esc düyməsinə basdıqda proqram bağlanacaq. Bunun səbəbi, Esc düyməsi yoxlanılmazdan əvvəl mausun x dəyərinin 250-dən böyük olub-olmamağını yoxlamağımızdır. Proqramımızın eni 500 piksel ölçüsündədir. 250 dəyəri isə proqram pəncərəsinin tam ortasını göstərir. Mausun oxu proqram pəncərəsinin sağ tərəfinə keçmiş olduqda x dəyəri 250-dən böyük olur.

İndi isə x əvəzinə y dəyərini yoxlayaq.

```

void keyboard(unsigned char key, int x, int y){
    if(y>250){
        if(key==27){
            exit(0);
        }
    }
}

```

Bu dəfə isə, mausun oxu proqram pəncərəsinin aşağı hissəsində olduqda proqram bağlanacaq, yuxarı hissəsində olduqda isə bağlanmayacaq. Çünki y dəyəri

yuxarıdan aşağıya doğru 0-dan 500-ə qədər artmaqdadır. Buna görə də, mausun oxu aşağıda olduqda y dəyəri 250-dən böyük olmuş olur.

Maus əməliyyatları. Mausun klik funksiyası.

Aşağıdakı kodu nəzərdən keçirək.

```
#include <GL/glut.h>

void mouse(int button, int state, int x, int y)

{
    exit(0);
}

void display() {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Game");
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}
```

Burada mouse adlı funksiyamız var və 4 parametr almaqdadır. Birinci parametr mausun düyməsi, ikinci parametr düymənin vəziyyəti, üçüncü parametr x dəyərini, dördüncü parametr isə y dəyərini göstərir. Proqramımızın main hissəsində isə, **glutMouseFunc(mouse);** funksiyası ilə mouse funksiyamız proqrama daxil edilir. Proqramı işə saldıqdan sonra hər-hansı bir maus əməliyyatında proqram bağlanacaq. Sol, orta və ya sağ düyməyə klik etdikdə **exit(0);** funksiyası vasitəsilə proqramımız bağlanacaqdır. Bunun səbəbi, mouse funksiyasında heç bir şərt qoymadan, düyməni, vəziyyəti, koordinatı təyin etmədən birbaşa əməliyyatı yazmağımızdır.

Maus funksiyamızı aşağıdakı şəkildə yazaq.

```
void mouse(int button, int state, int x, int y)
{
    if(button==GLUT_LEFT_BUTTON){
        exit(0);
    }
}
```

Burada **if(button==GLUT_LEFT_BUTTON)** şərti ilə əməliyyat aparılan düymənin sol düymə olub-olmadığını yoxlayırıq. Beləliklə, yalnız mausun sol düyməsinə klik etdikdə şərt ödənəcək və proqramımız bağlanacaq.

GLUT_LEFT_BUTTON – sol düyməni bildirir

GLUT_MIDDLE_BUTTON – orta düyməni bildirir

GLUT_RIGHT_BUTTON – sağ düyməni bildirir

Bu dəyərlər vasitəsilə, əməliyyat aparılan düymənin sağ, sol, yoxsa orta düymə olub-olmadığını yoxlaya bilərik.

Düymənin hansı düymə olmağını yoxladıqdan sonra əməliyyatın tərzini də müəyyən edə bilərik. Məsələn, biz əməliyyatı istifadəçi düyməyə basan anda edə bilərik. Bundan əlavə əməliyyatı istifadəçi düyməyə basandan sonra düyməni buraxdıqdan sonra edə bilərik. Funksiya daxilindəki ikinci parametr olan state parametri vasitəsilə bunu edə bilərik. Aşağıdakı koda baxaq.

```
#include <GL/glut.h>

#include <GL/glu.h>

void mouse(int button, int state, int x, int y)

{

if(state==GLUT_UP){

exit(0);

}

}

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 0.0);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");
```

```

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutMainLoop();

return 0;

}

```

Burada proqramın əvvəlində iki kitabxana əlavə etdiyimizə diqqət edək, birincisi **#include <GL/glut.h>**, ikincisi isə **#include <GL/glu.h>** kitabxanasıdır. İndi isə proqramı işlətdikdən sonra mausun düyməsini basılı saxlayaq və düyməni buraxaq. Burada görəcəyik ki, düymə basılanda yox, düymə basıldıqdan sonra buraxılanda əməliyyat yerinə yetiriləcək, yəni, pəncərə bağlanacaq. Bunun səbəbi funksiya daxilində **if(state==GLUT_UP)** şərtini yazmağımızdır. Şərt daxilindəki state dəyişəni GLUT_UP qiymətini aldıqda bu onu göstərir ki, əməliyyat düymə basıldıqda yox, düymə basılıb buraxıldıqdan sonra yerinə yetiriləcək. **GLUT_UP** yerinə **GLUT_DOWN** yazıldıqda isə, düymə basılan kimi əməliyyat yerinə yetiriləcək, pəncərə bağlanacaq.

İndi isə funksiyaımızı aşağıdakı şəkildə dəyişək.

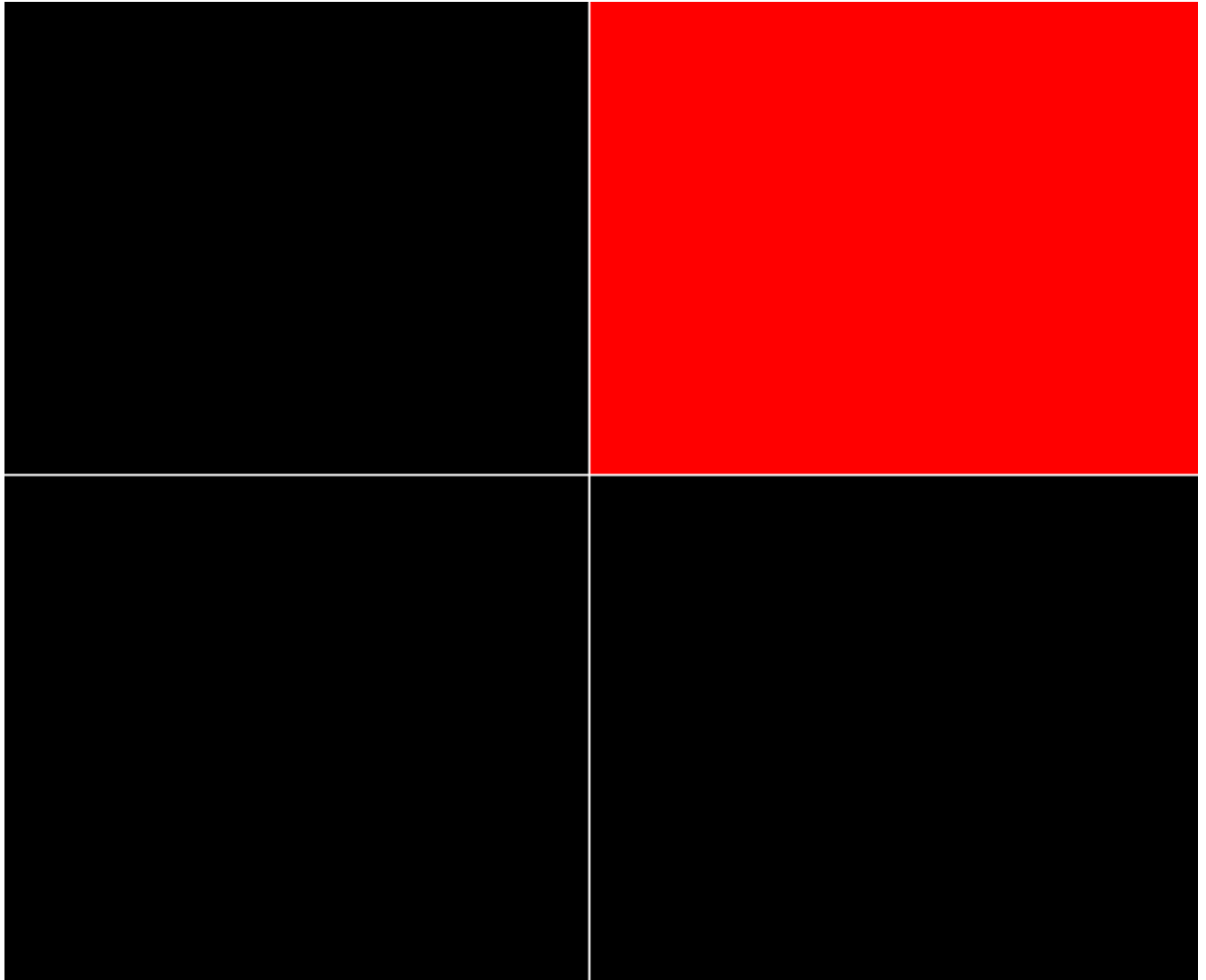
```

void mouse(int button, int state, int x, int y)
{
    if(button==GLUT_LEFT_BUTTON && state==GLUT_UP && x>250 && y<250){
        exit(0);
    }
}

```

Burada artıq yalnız pəncərənin yuxarisında, sağ tərəfdə, mausun sol düyməsi basılıb buraxıldıqda əməliyyat yerinə yetəcək, yəni, pəncərə bağlanacaq. Burada $x > 250$ olduğuna görə yalnız proqramın sağ tərəfində əməliyyatlar yerinə yetirilir. Eyni zamanda, $y < 250$ olduğuna görə, yalnız proqramın yuxarı hissəsində

əməliyyatlar yerinə yetirilir. Burada y yuxarıdan aşağıya doğru 0-dan 500-ə qədər artdığına görə, $y < 250$ şərti yuxarı hissəsini göstərir. Burada həm $x > 250$ həm də $y < 250$ şərtlərini yazdığımıza görə əməliyyatlar yalnız mausun oxu proqram pəncərəsinin yuxarı sağ tərəfində olduğu zaman yerinə yetiriləcək.



Əməliyyatlar yalnız mausun oxu qırmızı ilə işarələnmiş sahədə olduğu zaman baş verəcək.

Maus əməliyyatları. Mausun hərəkət funksiyası.

Bundan əvvəlki mövzuda, mausa klik etdikdə müxtəlif əməliyyatların aparılmasına baxdıq. İndi isə maus hərəkət etdirildikdə müxtəlif əməliyyatların aparılmasına baxaq. Zuma oyununu xatırlayaq, maus hərəkət etdirildikdə mərkəzdəki qurbağanın baxdığı istiqamət də dəyişir. OpenGL vasitəsilə də bu tip əməliyyatlar apara bilərik. Bunun **glutPassiveMotionFunc** funksiyasından istifadə edəcəyik. Aşağıdakı koda baxaq:

```
#include <GL/glut.h>

#include <stdio.h>

int yazi;

char str[10];

void funksiya(int x,int y){

yazi=x;

glutPostRedisplay();

}

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glRasterPos2f(0.0, 0.0);

sprintf(str,"%d",yazi);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, str);

glFlush();

}

int main(int argc, char** argv) {
```

```

glutInit(&argc, argv);

glutInitWindowSize(600, 600);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutPassiveMotionFunc(funksiya);

glutMainLoop();

return 0;

}

```

Gördüyümüz kimi, glutPassiveMotionFunc(funksiya); yazılaraq funksiya adlı funksiya mausun hərəkəti üçün çağırılır. Çağırılan funksiya baxaq.

```

void funksiya(int x,int y){

yazi=x;

glutPostRedisplay();

}

```

Burada iki parametr var. Birinci x parametri mausun oxunun olduğu nöqtənin x dəyərini, y parametri isə y dəyərini göstərir. Burada göstərilən qiymət - 1 və 1 arasında yox, ekranın eni 600 olduğu üçün 0 və 600 arasında olur. Yuxarıda yazdığımız proqramda, mausun oxunun yerini dəyişdikcə, proqram pəncərəsinin mərkəzində x dəyəri daimi olaraq dəyişərək göstərilir. Yuxarıdakı nümunədə printf funksiyasından istifadə edərək int tipli yazi dəyişənini char tipli chr dəyişən çoxluğuna köçürürük. Bunun səbəbi, glutBitmapString funksiyasındakı yazının char tipli olmasıdır.

Tam Ekran

İşlətdiyimiz bir çox oyunlar tam ekran formasındadır, yəni oyun pəncərəsi tam ekranı tutmaqdadır. Opengl işlədərək yazdığımız proqramı tam ekran etmək üçün **glutFullScreen()**; funksiyasından istifadə edə bilərik. Beləliklə proqramımız tam ekran olaraq bütün ekranı tutacaq. Aşağıdakı kodda bu funksiyanın işlənməsinə baxaq.

```
#include <GL/glut.h>

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glColor3f(0,255,0);

glRasterPos2f(0.0, 0.0);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, "Opengl");

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutFullScreen();

glutMainLoop();

return 0;
```

}

Beləliklə, proqramımız tam ekran olaraq bütün ekranı tutmaqdadır.

Zaman aralığı ilə əməliyyatlar

Öyrənməli olduğumuz vacib əməliyyatlardan biri də zaman əməliyyatlarıdır. Bir çox proqramlaşdırma mühitində olduğu kimi OpenGL daxilində də əməliyyatı bir neçə müddət sonra yerinə yetirmək mümkündür. Əməliyyatı bir neçə saniyədən sonra yerinə yetirmək üçün istifadə edəcəyimiz funksiya **glutTimerFunc()** funksiyasıdır. Aşağıdakı proqrama baxaq.

```
#include <GL/glut.h>

void interval(){

exit(0);

}

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutTimerFunc(3000,interval,0);

glutMainLoop();
```

```
return 0;  
  
}
```

Kodu yazıb proqramı kompilyasiya edək. Proqramı açdıqdan 3 saniyə sonra proqram bağlanacaq. İndi isə yazdığımız kodu izah edək. İlk öncə yazdığımız `interval()` funksiyasına baxaq.

```
void interval(){  
  
    exit(0);  
  
}
```

Gördüyümüz kimi, yazdığımız `interval` funksiyası çox sadde bir funksiyadır. Funksiyanın məqsədi, funksiya çağırıldıqda proqramı bağlamaqdır. İndi isə **`glutTimerFunc`** funksiyasına baxaq.

```
glutTimerFunc(3000,interval,0);
```

Burada birinci parametrimiz 3000-dir. Yəni, 3000 millisaniyə sonra qeyd edilmiş funksiya çağırılacaq. İkinci parametrimiz isə `interval`-dir. Qeyd etdiyimiz `interval` isə, yuxarıda yazdığımız funksiyanın adıdır. Bu onu göstərir ki, 3 saniyə sonra `interval` funksiyası çağırılacaq və onun əməliyyatları yerinə yetiriləcək. Nəticədə 3 saniyə sonra proqramımız bağlanacaq. Bu funksiya vasitəsilə əməliyyat yalnız bir dəfə qeyd etdiyimiz müddətdən sonra işə düşür. İndi isə aşağıdakı proqrama baxaq:

```
#include <GL/glut.h>  
  
float x=0.3;  
  
void interval(){  
  
    x=x+0.01;
```

```

glutPostRedisplay();

glutTimerFunc(100,interval,0);

}

void display() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_MODELVIEW);

glBegin(GL_LINES);

glVertex2f(0.2,0.5);

glVertex2f(x,0.5);

glEnd();

glLoadIdentity();

glFlush();

}

int main(int argc, char** argv) {

glutInit(&argc, argv);

glutInitWindowSize(900, 800);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutTimerFunc(100,interval,0);

glutMainLoop();

return 0;

}

```

Kodu yazıb proqramı kompilyasiya edək. Proqramı açıqda hər 100 millisaniyədən bir xəttin uzandığını görəcəyik. İndi isə kodumuzu izah edək.

İlk olaraq proqramımızın əvvəlində bir x global dəyişəni elan edib qiymətini 0.3 edirik. Elan etdiyimiz x dəyişəni çəkəcəyimiz xəttin bitmə nöqtəsinin x dəyərini göstərəcək. Xəttimizi çəkərkən xəttin son nöqtəsi üçün sabit bir nöqtə yazmaq yerinə x dəyişənini yazırıq.

```
glVertex2f(x,0.5);
```

Beləliklə xəttinin son nöqtəsinin x dəyəri x dəyişəninin qiymətini almış olur.

Növbəti olaraq **glutTimerFunc(100,interval,0)** funksiyası vasitəsilə 100 millisaniyə sonra bir dəfə üçün interval funksiyasını çağıraraq. İndi isə interval funksiyasına baxaq.

```
void interval(){
```

```
x=x+0.01;
```

```
glutPostRedisplay();
```

```
glutTimerFunc(100,interval,0);
```

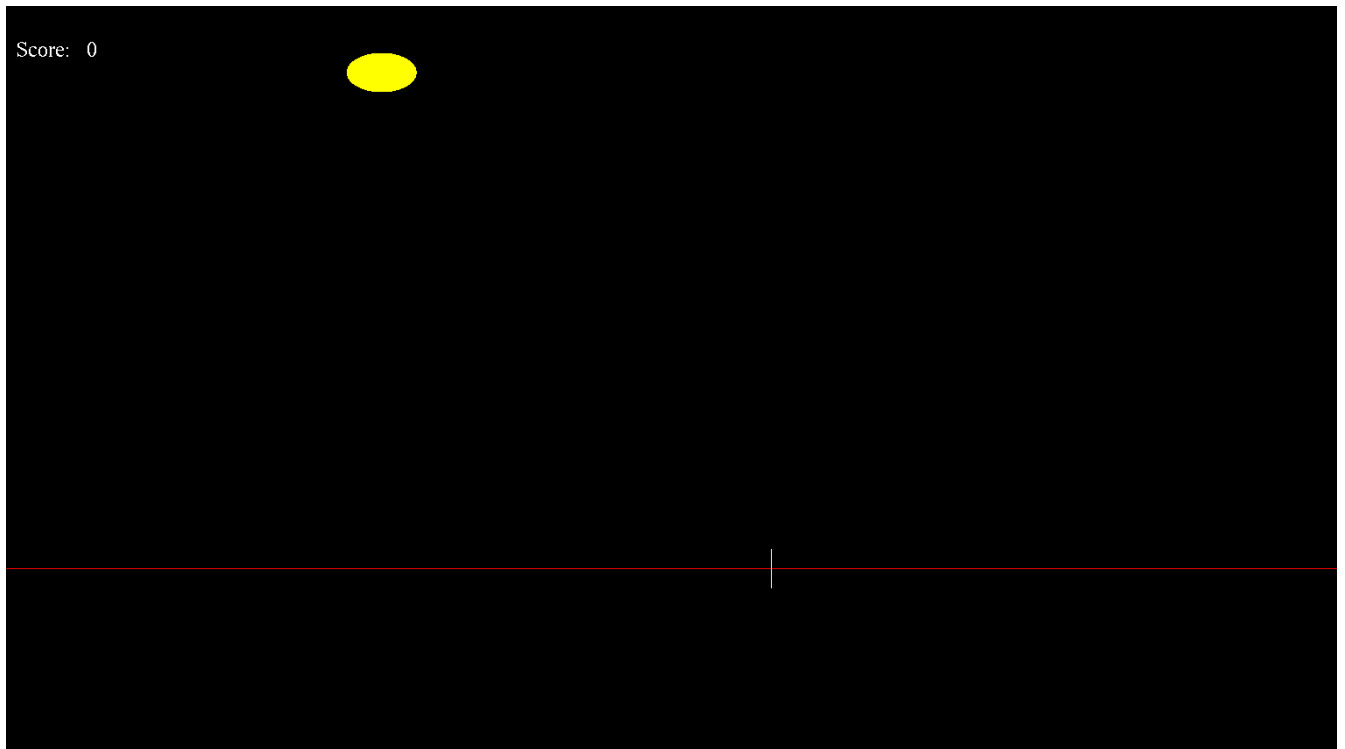
```
}
```

Funksiya çağırılarkən ilk olaraq x dəyişəninin dəyəri 0.01 qədər artır. Beləliklə, xəttimizin son nöqtəsinin x dəyəri artır, xətt uzanır. Növbəti olaraq ilk dəfə işlədəcəyimiz **glutPostRedisplay()** funksiyasını yazırıq. Adından da başa düşmək olar ki, bu funksiya edilmiş dəyişikliklərdən sonra səhifəni yeniləyərək dəyişiklikləri istifadəçiyə göstərmək üçündür. Əgər bu funksiyanı yazmasaq, əməliyyatlar əslində baş versə belə biz o dəyişiklikləri görməyəcəyik. Onun üçün də, funksiya sadə olsa da çox əhəmiyyətli bir funksiyadır. Növbəti olaraq isə, **glutTimerFunc(100,interval,0)** funksiyasını yenidən yazırıq. Deməli nə baş verəcək? İlk dəfə bu funksiyanı işlətdikdə cəmi bir dəfə üçün interval funksiyası 100 millisaniyə sonra çağırılacaq. Ancaq bu funksiya bir dəfə çağırır, bizə isə lazımdır ki, daimi olaraq çağırılsın. Ona görə interval funksiyasının sonunda, **glutTimerFunc** funksiyasından istifadə edərək intervalı yenidən çağırırıq. Nəticədə 100 millisaniyə sonra interval yenidən çağırılır. Bu dəfə də həmin proseslər baş verir və 100 millisaniyə sonra interval yenidən çağırılır. Beləliklə, sonsuz olaraq funksiya hər 100 millisaniyədən bir çağırılır və daimi əməliyyatlar baş verir.

Beləliklə, əsas funksiyaımız olan **glutTimerFunc** funksiyaından istifadə edərək zaman aralığı ilə bir çox əməliyyatları yerinə yetirə bilirik.

Kiçik oyunun yaradılması

İlk kiçik oyunumuzun görüntüsü aşağıdakı şəkildə olacaq.



Oyunda yuxarıdan aşağıya doğru gələn sarı topu, aşağıdakı qırmızı xəttə gəlib çatmadan ağ güllə vasitəsilə vurmalıyıq. Hər dəfə topu vurduqda, xalımız artır. Əgər sarı top qırmızı xəttə gəlib çatarsa, o zaman, xalımız sıfırlanır. İndi isə, hissə-hissə oyunun yazılmasını izah edək. İlk olaraq oyunumuzun fiqurlarını, görüntüsünü yaratmalıyıq. Aşağıdakı kodları yazaq:

```
#include <GL/glut.h>
```

```
#include <math.h>
```

```
#include <stdio.h>
```

```
float direction_x=0.0;
```

```
float direction_y=-0.45;
```

```
float ball_x;
```

```
float ball_y=0.9;
```



```

int yuxari=0;

int click=0;

int score=0;

char str[10];

double randBetween(double min, double max)
{
    return ((double)rand()/RAND_MAX) * (max - min) + min;
}

void drawCircle(float r, float x, float y) {
    float i = 0.0f;
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(x, y); // Center
    for(i = 0.0f; i <= 360; i++)
        glVertex2f(r*cos(M_PI/180.0 * i) + x, r*sin(M_PI/180.0 * i) + y);
    glEnd();
}

void display() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_LINES);
    glColor3f(255,0,0);
    glVertex2f(-1,-0.5);

```

```

glVertex2f(1,-0.5);

glEnd();

glBegin(GL_LINES);

glColor3f(255,255,255);

glVertex2f(direction_x,direction_y);

glVertex2f(direction_x,direction_y-0.10);

glEnd();

glColor3f(255,255,0);

drawCircle(0.05,ball_x,ball_y);

glColor3f(255,255,255);

glRasterPos2f(-0.9, 0.8);

sprintf(str,"%d",score);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, "Score: ");

glRasterPos2f(-0.8, 0.8);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, str);

glFlush();

}

int main(int argc, char** argv) {

srand(time(NULL));

ball_x=randBetween(-0.9,0.9);

glutInit(&argc, argv);

glutInitWindowSize(600, 600);

glutCreateWindow("Game");

glutDisplayFunc(display);

```

```
glutFullScreen();  
glutMainLoop();  
return 0;  
}
```

Aşağıdakı əmr vasitəsilə kodu kompilyasiya edək:

```
gcc goruntu.c -o goruntu -L/usr/X11R6/lib/ -lGL -lGLU -lglut -lm
```

Yazdığımız kod vasitəsilə oyunun görünüşünü hazırlamış olacağıq. İndi isə yazdığımız kodu izah edək. İlk öncə elan etdiyimiz dəyişənlərə baxaq.

```
float direction_x=0.0; //Güllənin, yəni ilk başda qırmızı xəttin üstündə olan ağ  
rəngli kiçik xəttin x nöqtəsini göstərir
```

```
float direction_y=-0.45; // Həmin xəttin y nöqtəsini göstərir
```

```
float ball_x; // Sarı topun x nöqtəsini göstərir
```

```
float ball_y=0.9; //Sarı topun y nöqtəsini göstərir
```

```
int yuxari=0; // Sarı topu vurmaq üçün düyməyə basdıqda bu dəyişənin qiyməti 1  
olacaq, hal-hazırda 0 qiymətini alır
```

```
int score=0; // İstifadəçinin yığdığı xalı göstərir
```

```
char str[10]; // İstifadəçinin yığdığı xalı göstərən ədəd tipli dəyişən bu dəyişənə  
yazı olaraq köçürülür
```

Dəyişənlərimizi izah etdik. İndi isə digər hissələrə baxaq.

Yazdığımız `randBetween()` funksiyası – İki ədəd aralığında təsadüfi ədəd seçmək üçün istifadə edəcəyik. Sarı top yarandıqda x koordinatı təsadüfi olaraq təyin olunur. Yazının yazılmasına baxaq.

```

glRasterPos2f(-0.9, 0.8);

sprintf(str,"%d",score);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, "Score: ");

glRasterPos2f(-0.8, 0.8);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, str);

```

Bu hissədə istifadəçinin topladığı xal yazılır. Sprintf funksiyası vasitəsilə score dəyişənindəki xal str dəyişəninə köçürülərək glutBitmapString funksiyası vasitəsilə ekrana yazılır.

Görüntünü yaratdıqdan sonra içərisində hərəkətləri də əks etdirən oyunun bütün kodlarını yazaq.

```

#include <GL/glut.h>

#include <math.h>

#include <stdio.h>

float direction_x=0.0;

float direction_y=-0.45;

float ball_x;

float ball_y=0.9;

int yuxari=0;

int score=0;

char str[10];

double randBetween(double min, double max)

{

```

```

    return ((double)rand()/RAND_MAX) * (max - min) + min;
}

void interval(){
    ball_y-=0.01;
    if(yuxari==1){
        direction_y+=0.05;
        if(direction_x<=(ball_x+0.05) && direction_x>=(ball_x-0.05)){
            if(direction_y>=(ball_y+0.05)){
                srand(time(NULL));
                ball_x=randBetween(-0.9,0.9);
                ball_y=0.9;
                direction_y=-0.45;
                yuxari=0;
                score++;
            }
        }
        if(direction_y>1){
            direction_y=-0.45;
            yuxari=0;
        }
    }
    if(ball_y<=-0.495){
        ball_y=0.9;
        score=0;
    }
}

```

```

}

glutPostRedisplay();

glutTimerFunc(32,interval,0);

}

void drawCircle(float r, float x, float y) {

float i = 0.0f;

glBegin(GL_TRIANGLE_FAN);

glVertex2f(x, y); // Center

for(i = 0.0f; i <= 360; i++)

glVertex2f(r*cos(M_PI/180.0 * i) + x, r*sin(M_PI/180.0 * i) + y);

glEnd();

}

void mouse(int x, int y){

if(yuxari==0){

direction_x=x/300.0f-1.0;

}

glutPostRedisplay();

}

void mouse_click(int button, int state, int x, int y){

if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {

yuxari=1;

}

}

void keyboard(unsigned char kod,int x,int y){

```

```

if(kod==27){
    exit(0);
}
}

void display() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_LINES);
    glColor3f(255,0,0);
    glVertex2f(-1,-0.5);
    glVertex2f(1,-0.5);
    glEnd();
    glBegin(GL_LINES);
    glColor3f(255,255,255);
    glVertex2f(direction_x,direction_y);
    glVertex2f(direction_x,direction_y-0.10);
    glEnd();
    glColor3f(255,255,0);
    drawCircle(0.05,ball_x,ball_y);
    glColor3f(255,255,255);
    glRasterPos2f(-0.9, 0.8);
    sprintf(str,"%d",score);

```

```

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, "Score: ");

glRasterPos2f(-0.8, 0.8);

glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, str);

glFlush();

}

int main(int argc, char** argv) {

srand(time(NULL));

ball_x=randBetween(-0.9,0.9);

glutInit(&argc, argv);

glutInitWindowSize(600, 600);

glutCreateWindow("Game");

glutDisplayFunc(display);

glutPassiveMotionFunc(mouse);

glutSetCursor(GLUT_CURSOR_NONE);

glutKeyboardFunc(keyboard);

glutMouseFunc(mouse_click);

glutTimerFunc(32,interval,0);

glutFullScreen();

glutMainLoop();

return 0;

}

```

İndi isə yazdığımız kodu hissə-hissə izah edək. İlk olaraq mausun hərəkət funkiyasına baxaq.


```

void mouse(int x, int y){
    if(yuxari==0){
        direction_x=x/300.0f-1.0;
    }
    glutPostRedisplay();
}

```

Bu mausun hərəkət funksiyasıdır. Burada iki ölçüdən istifadə olunur. Funksiyanın parametri olan x dəyəri 0 və 600 arasında dəyişir. Oxun koorindatları isə -1 və 1 aralığında verilir. Buna görə də, bu dəyər 300-ə bölünərək 1 çıxılır. Növbəti olaraq mausun klik funksiyasına baxaq.

```

void mouse_click(int button, int state, int x, int y){
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        yuxari=1;
    }
}

```

Əgər mausun sol düyməsinə basılırsa, yuxarı dəyişəni 1 olur. Bu o deməkdir ki, istifadəçi gülləni yuxarıya doğru atmaq istəyir.

Keyboard funksiyamız isə sadədir, sadəcə esc düyməsinə basıldıqda proqramnan çıxılır.

Son olaraq interval funksiyasına baxaq.

```

void interval(){
    ball_y-=0.01;
}

```

```

if(yuxari==1){
direction_y+=0.05;
if(direction_x<=(ball_x+0.05) && direction_x>=(ball_x-0.05)){
if(direction_y>=(ball_y+0.05)){
srand(time(NULL));
ball_x=randBetween(-0.9,0.9);
ball_y=0.9;
direction_y=-0.45;
yuxari=0;
score++;
}
}
if(direction_y>1){
direction_y=-0.45;
yuxari=0;
}
}
if(ball_y<=-0.495){
ball_y=0.9;
score=0;
}
glutPostRedisplay();
glutTimerFunc(32,interval,0);
}

```

Sarı topumuz daima aşağı enəcəyinə görə, funksiyanın əvvəlində `ball_y=0.01;` yazılır. Əgər düyməyə klik olunubsa, yəni `yuxari` dəyişəni 1-ə bərabərdirsə, o zaman, `direction_y+=0.05` oxu yuxarıya doğru aparırıq. Aşağıdakı iki şərtə baxaq.

```
if(direction_x<=(ball_x+0.05) && direction_x>=(ball_x-0.05)){  
    if(direction_y>=(ball_y+0.05)){  
        srand(time(NULL));  
        ball_x=randBetween(-0.9,0.9);  
        ball_y=0.9;  
        direction_y=-0.45;  
        yuxari=0;  
        score++;  
    }  
}
```

Bu şərtlər ödənirsə, deməli güllə topa toxunmuşdur. O zaman, yenidən topu təsadüfi olaraq bir yerə yerləşdiririk və istifadəçinin xalını artırırıq. Bir dəfə klik olunduqdan sonra yenidən `yuxari` dəyişənini 0 edirik.

Aşağıdakı şərtə baxaq:

```
if(direction_y>1){  
    direction_y=-0.45;  
    yuxari=0;  
}
```

Əgər oxu düzgün atmamışıqsa və topa dəymirsə, pəncərənin yuxarisına çatdıqdan oxu yenidən aşağıya yerləşdiririk.

```
if(ball_y<=-0.495){  
    ball_y=0.9;  
    score=0;  
}
```

Yuxarıdakı şərtə isə, əgər sarı top aşağıdakı qırmızı xəttə toxunarsa, o zaman topu yenidən yuxarı qaytarırıq və oyunçu məğlub olduğu üçün onun xalını sıfırlayırıq.

Oyunda əlavə olaraq aşağıdakı funksiya da istifadə etmişik:

```
glutSetCursor(GLUT_CURSOR_NONE);
```

Bu funksiya mausun oxunu oyunumuzda gizlətmək üçündür.

Beləliklə oyunumuz hazır olmuş oldu.

Kitabın sonu

Bu buraxılışa əsas olaraq OpenGL mövzuları əlavə olunmuşdur. Mövzular vasitəsilə kiçik ikiölçülü proqramlar, oyunlar hazırlamaq mümkündür. Növbəti buraxılışda əsas olaraq üçölçülü qrafikaya üstünlük verilməsi nəzərdə tutulur.

Hələlik bu qədər, növbəti buraxılışda görüşənə qədər :)