Rasim Aşurov

birinci buraxılış

Başlayanlar üçün

C++

Bu kitabı oxuyub siz:

- Proqramlaşdırmanın nə olduğunu anlayacaqsınız
- C++ dilində proqramlar yazmağı öyrənəcəksiniz
- Obyektli proqramlaşdırmanın əsasları anlayacaqsınız
- Java və C# dillərini öyrənməyə olduqca asan olacaq

Rasim Aşurov

C++ qorxusuz

Bakı, 2010-cu il.

Müəllif haqqında...

Aşurov Yasin oğlu Rasim 1992-ci ilin yanvarın 27-sində anadan olub. Hələ uşaq ikən kompüterə böyük həvəslə yanaşan Rasim, artıq 14 yaşından kompüteri öyrənməyə can atmışdır. 15 yaşında Windows əməliyyat sistemlərini və ümumdünya İnternet strukturunu mənimsədikdən sonra proqramlaşdırma dillərini öyrənməyə başlamışdır. 2008-ci ildə Bakı Kompüter Kollecinə daxil olmuşdur. Artıq 17 yaşında 2 proqramlaşdırma dillərinə yiyələnmiş Rasim, sonralar kitab yazmaq istedadına nail olmuşdur. Qısa zaman ərzində 2 kitab yazmışdır. Onlardan "Mənim mənimsədiklərim, bildiklərim, bacardıqlarım..." və nəhayət "C++ qorxusuz". Əsas məqsədi:

"Bu həyatda öz yerini tapmaq, professional proqramçı olmaq və irəliyə getməkdir."

Müəllifdən...

Gündən günə proqramlaşdırmanı öyrənmək istəyən adamlar çoxalır. Onlar müxtəlif öyrədicilər (öyrədici jurnal-kitablar) əldə edirlər (alırlar, internetdən yükləyirlər). Lakin təəssüf ki, heç bir nəticə əldə etmirlər. Nəyə görə? Bu sualın cavabları çoxdur:

- ✓ *proqramlaşdırmanın nə olduğunu bilmirlər*
- ✓ *proqramlaşdırmaya uyğun giriş yoxdur*
- ✓ *hansı dildən başlanmasını bilmirlər, nəticədə özlərini yarı yolda azdırırlar.*

Proqramlaşdırma dilləri çoxdur. Hərəsinin özünə görə yaxşı və pis cəhətləri var. Lakin bütün proqramlaşdırma dillərin (HTML, XML,WML və s. dilləri istisna olmaqla, bu dillər qipermətnin ölçü dilləri sayılır və proqramlaşdırma dillərinə heç bir aidiyyəti yoxdur) kökü ən aşağı səviyyəli, maşın dili (olan) hesab olunan Assembler dilidir. Hər bir proqramçı bu dilə hörmət etməlidir (lakin onu öyrənmək vacib deyil!). Hörmət deyəndə nə başa düşülür? Elə proqramçılar (və yaxud sadəcə proqramlaşdırma dillərini bilən adamlar) var ki, onlar bəzi dillərin çətin olduğuna görə onlara nifrət edir, onlardan çəkinir. Bu düzgün deyil. Hər bir adam ona uyğun olan proqramlaşdırma dili öyrənir, onların vasitəsilə veb-proqramlar, proqramlar və yaxud script-lər yazır. Ona görə də proqramçının heç bir dildən zəhləsi getməməlidir. Misal üçün, C++ dili Delphi dilinə nəzərən çox çətin hesab olunur. Ona görə də C++ dili təmiz kişi cinsi üçün nəzərdə tutulub. Qadınlar arasında C++ dilini öyrənmək istəyənlər də var, lakin onların məntiqi buna imkan vermir. Və bu qadınlar C++ dilinə nifrətlə yanaşmağa başlayırlar. Qadınlar Delphi dilini öyrənə bilsə də, C++ dilini öyrənmək onlar bacarmazlar, çünki bu dil qadınlar üçün nəzərdə tutulmayıb (lakin elə qadınlar arasında elələri də var ki, onların məntiqi kişi cinsinə nəzərən qat-qat üstündür). Qadınların C++ dilinə nifrəti düzgün deyil. Burada mən qadınları misal kimi götürmüşəm, onların yerinə C++ dilini öyrənməyə bacarmayan kişi cinsi də ola bilər.

Beləliklə, mən – Rasim, bunların hamısını nəzərə alaraq, C++ dilinin öyrədici kitabını doğma dilimiz olan azərbaycan dilində yazdım. Mən bu kitabı elə-belə yazmamışam, bu kitabı yazmaqda mən böyük əziyyət çəkmişəm, vaxt sərf etmişəm. Lakin yəqin edirəm və həm də əminəm ki bu əziyyətim hədər yerə getməyib. Sizə C++ dilini öyrənməkdə uğurlar arzulayıram.

Mən həmişə fəxr etmişəm, edirəm və edəcəm – Azərbaycan mənim vətənimdir!

Rasim Aşurov Y.
Bakı, 28 Aprel 2009.

Təşəkkürlərim...

Təşəkkür etmək istədiklərim adamlar çoxdur. Lakin ilk öncədən atama, anama, bacıma böyük təşəkkürlərimi bildirirəm. Çünki onlar mən bu kitabı yazarkən məni həvəsləndirirdilər. Sonra, hal-hazırda təhsil aldığım Bakı Kompüter Kollecində olan Kiçik Elmlər akademiyasına, ən əsası, bu akademiyanın təşkilatçısı olan Nəcibə müəlliməyə böyük təşəkkürlərimi bildirirəm. Nəcibə müəllimə məni tanınmazlıq bataqlığından çıxartdı və bütün kollecə tanıtdırdı, mənimsədiklərimə, bildiklərimə, bacardıqlarıma qiymət verməyi məcbur etmişdir. Həm də ki mənə, mənim əməyimə hörmət edənlərə böyük təşəkkürlərimi bildirirəm...

Müəllif ilə əlaqə

Bir çox oxucular kitabın rəylərini mənə göndərməkdən çəkinirlər. Mən sizi çox yaxşı başa düşürəm, çünki bilirəm ki, siz elə fikirləşirsiniz ki, əgər mənim işlərim çoxdursa onda məni narahat etməyə dəyməz. Lakin mən sizin əksinə deyirəm. Mənim elektron poçtuma məktub göndərməklə - kitab haqqında fikirləri söyləməklə siz məni narahat etmirsiniz, əksinə mən oxucuların mənim kitablarım haqqında fikirlərini bilməklə çox sevinirəm və bunu özümə dərs hesab edirəm. Yəni hazırki kitabın çatışmamazlıqlarından istifadə etməklə gələcəkdə yazacağım kitablarda bunları nəzərə alacam və ktiablarım yaxşı olmağına doğru can atacam. Beləliklə siz mənimlə aşağıdakı üsullarla əlaqə saxlaya bilərsiniz:

E-mail address: i80586@mail.ru
ICQ: 488-894-309

Kitabın xüsusiyyətləri

Çoxlu sayda C++ haqqında kitablar mövcuddur. Lakin onların çoxları bizi qane etmir, yəni o kitablar artıq “C” dilini bilənlər üçün nəzərdə tutulmuşdur. Lakin bu kitabda mən bunların hamısını nəzərə almışam. Sizdən ancaq – bir az Windows əməliyyat sistemini bilmək, proqramları (misal üçün mətn redaktoru) başlatmaq bacarığı tələb olunur.

Bu kitab haqqında daha nə söyləmək olar?

Əlinizdə saxladığınız kitab proqramlaşdırmanın nə olduğunu söyləyir, proqram yazmaq prosesini aydınlaşdırır. Əgər siz nə vaxtsa proqramlaşdırmanı öyrənmişdinizsə (misal üçün kollecdə, universitetdə və s.) və yaxud proqramlaşdırma ilə məşğul olurdunuzsa, onda bu daha da yaxşı. Çünki bu kitab vasitəsilə siz biliklərinizi daha da möhkəmləndirə bilərsiniz.

Əgər siz C++-dan fərqli proqramlaşdırma dilini bilirsiniz, onda nə?

Əgər siz digər bir proqramlaşdırma dilini bilirsinizsə, C++-dan isə başınız çıxmırsa, bu problem deyil. Əsası odur ki məntiq olsun. Həm də proqramçı kimi fikirləşməyi bacarmaqdır.

Niyə məhz C++ dilindən başlamaq lazımdır?

Çox adamlar sizə deyə bilərlər ki, C++ dili başlayan adamlar üçün deyil və təcrübəli proqramçı səviyyəsinə çatmamağına qədər ondan heç fikirləşmək də lazım deyil. Mən bununla razı deyiləm.

Sizin proqramçı karyeranıza başlamazdan əvvəl C++ dilini nə üçün bilmək lazım olduğunun bir neçə səbəbləri var. Adamlar “C” dilini bilmək üçün çoxlu əziyyətlər çəkmişdirlər. Lakin “C” dili real işdə az istifadə olunur. İndi tələbələr bu dili C++ dilinə keçmək üçün öyrənirlər. Lakin bu mənasızdır. “C” dilini öyrənməklə bəzi pis vərdişlər qala bilər. Ondan yaxşısı elə birbaşa C++ dilinə keçməkdir, yəni onu öyrənməkdir. C++ dili indi həm sistem proqramçıları üçün, həm də kommersial proqram təminatı yazan proqramçılar üçün nəzərdə tutulub.

Bir neçə digər dillər (misal üçün Microsoft kompaniyasından Visual Basic) işə yaramır. Basic də “C” dili kimi pis vərdişlər saxlaya bilər. C++ dili onu öyrənənə hədiyyə kimi bir çox nəticələr verir.

- ✓ “C” dili kimi, C++ dili də sistem proqramlaşdırma (systems-programming language) dili adlanır.
- ✓ “C” dilindən fərqli olaraq C++ dili obyekti dil hesab olunur.

Kitabın birinci yarısında C++ dilinin fundamental əsasları yazılıb: necə proqramı yaradaq və onun işləməsini təmin edək.

BAŞLIQ 1

Sizin C++ dilində ilk proqramlarınız

Əslində C++ proqramlaşdırma dilində qorxulu heç bir şey yoxdur. Digər proqramlaşdırma dilləri kimi, C++ da kompüterə məntiqi dəqiq funksiyalar göndərmək üsuludur. C++ dili sizin istədiyiniz qədər çətin ola bilər, lakin əvvəla onu, fundamental əsasları yazmaq üçün istifadə etmək lazımdır. Bu kitabın girişi belədir.

Birinci bölmələrdə mən proqramlaşdırma əsasları haqqında deyəcəm. Lakin əgər siz artıq proqramlaşdırma ilə məşğul olmusunuzsa, onda siz bu bölmələri oxumayıb keçə və yaxud tez izləyə bilərsiniz.

Proqramçı kimi fikirləşmək

Proqramlaşdırma digər bir işə oxşaya da bilər, hansı ki siz nə vaxtsa məşğul olmusunuz. Əsası siz burada instruksiyalar verirsiniz, hansı ki məntiqi yolla qurulur (sizin beyninizdə).

Kompüterlər yalnız biz istədiyimiz işləri görürlər.

Kompüterlər yalnız bizim onlara dediyimiz işi görürlər: bu cümlə bu kitabın əsasıdır - həm də ki əgər siz proqramlaşdırmada yenisinizsə. Proqramlaşdırma dilindən istifadə etməklə, misal üçün C++, Visual Basic, Pascal və yaxud Fortran, siz kompüterə prosedurlar sırasını verirsiniz, hansı ki onlar yerinə yetiriləcək. Elə bu prosedurlar sırası – proqramdır.

Müəyyən edin ki, proqram nə yerinə yetirsin

Belə, kompüterin nəsə yerinə yetirməsi üçün ona dəqiq demək demək lazımdır ki, o nə etsin.

Bu dövrə qədərki dövrə, siz proqramlar istifadə etmisiniz, hansı ki digər proqramçılar (misal üçün Bill Gates) tərəfindən sizin üçün yazılmışdır. Belə keyfiyyətdə siz son istifadəçilər, axırıncı nəticədə sadəcə istifadəçilər olmusunuz.

İndi, nə vaxt ki siz özünüz proqramlar yazacaqsınız, siz daha yüksək mərhələdə - proqramçı olacaqsınız. İndi siz özünüz müəyyən edəcəksiniz – proqram nə yerinə yetirsin.

C++ dili üçün ekvivalent instruksiyaların yazılışı

Proqramın addım-addım nə yerinə yetirdiyini qeyd edəndən sonra onun ekvivalent instruksiyalarını yazmaq lazımdır.

Misal üçün, biz istəyirik ki proqram aşağıdakıları yerinə yetirsin:

“Mən C++ dilini öyrənirəm” cümləsini göstərsin (MS-DOS sətrlərində !).

Bunu etmək üçün aşağıdakı kodu yazırıq:

cout << “Men C++ dilini oyrenirem” ;

Yadda saxlayın ki, proqramlaşdırmanın əsasını, kompüterin nəsə yerinə yetirməyi təşkil edir. Lakin kompüter yalnız öz dilini – maşın kodunu başa düşür. 1950-ci illərdə proqramçılar instruksiyalarını elə bu maşın dilində yazmışdırlar və bu proses çox çətin və çox vaxt tələb edən idi.

Proqramların yazılmasını asanlaşdırmaq üçün mütəxəssislər Fortran, Basic, C kimi proqramlaşdırma dillərini istehsal ediblər, hansı ki maşın dilini bilmədən proqram yazmağını təmin edirdi.

Proqramı yazmaq üçün siz **pseudocode**-dən istifadə edə bilərsiniz. Aşağıda görsədilən misal elə bununla yazılıb (yəni ana dilində):

əgər (if operatoru**) a b-dən böyükdürsə**
    onda
yazmaq (print operatoru) **“a b-dən böyükdür.”**
    əks halda
yazmaq (print operatoru) **“a b-dən böyük deyil.”**

Belə, psevdokodu (ana dilini) yazandan sonra, C++ dilində yazılan proqramın alınmasına bir az qalıb. Qalıb ki, hər bir addımın C++ dilində uyğun instruksiyasını tapmaq və yazmaq.

if (*a* > *b***)**
    cout << **“***a b-dən böyükdür***.”** ;
else
    cout << *“a b-dən böyük deyil.”*;

C++ dilinin instruksiyaları o qədər dəqiqdir ki, sıfırlar və birlərdən ibarət maşın koduna çevrilsin.

Qəribə olmamalıdır ki, proqramlaşdırma dillərinin ciddi sintaksis qaydaları var. Misal üçün if-else instruksiyasının sintaksisi:

if (*şərt*)
    instruksiya
else
    instruksiya

Qalın olan sözlər açarlı sözlər (keywords) adlanır; onlar proqrama necə yazıldığı şəkildə əlavə olunmalıdırlar. Kursiv ilə seçilmiş sözər isə instruksiyalar daşıyır, hansı ki siz onların yerinə əlavə edəcəksiniz.

C++ dilinin instruksiyalarını (və yaxud kodunu) maşın koduna çevirən proqram isə *kompilyator* (compiler) adlanır.

Bir neçə terminlər

Proqram yazmağa başlamazdan qabaq siz bir çox şeyləri bilməlisiniz. Mən aşağıda bunları qeyd etmişəm.

Prilojeniya (application)

Fiziki cəhətdən prilojeniya da proqram kimi bir şeydir, lakin istifadəçi nöqtəyi nəzərdən. Prilojeniya – proqramdır, hansı ki hər hansı bir funksiyanı yerinə yetirmək üçün başladılır. Mətn redaktoru – prilojeniyadır; Veb-bələdçi – prilojeniyadır. Həm də kompilyator prilojeniya hesab olunur, çünki proqramçılar tərəfindən istifadə olunur. Qısa desək, proqram yazılandan, testlənəndən sonra prilojeniya hesab olunur.

Kod (code)

Kod – “proqram” sözünün sinonimidir, lakin proqramçının nöqtəyi nəzərdən. Proqramı yaratmaq üçün yazdığımız sözlərin toplusu “kod” adlanır.

Kompilyator (compiler)

Dil tərcüməçisi, hansı ki girişdə C++ dilinin kodunu alır və çıxışda maşın kodunda hazır proqram alınır. Belə tərcümə vacibdir, çünki kompüter – mərkəzi prosessor (CPU) yalnız maşın dilini başa düşür.

Maşın kodu (machine code)

Mərkəzi prosessorun öz şəxsi dili, hansı ki sıfırlardan və birlərdən ibarətdir. Maşın kodunda proqram yazmaq da olar, lakin bu çox uzun çəkir, həm də mərkəzi prosessor arxitekturasının biliyini tələb edir – hər iki sual kitabımızdan xaricdir.

Proqram (program)

Kommandalar sırasıdır, hansı ki yerinə yetiriləcək. Əvvəla qeyd etdiyim kimi, proqramı yazmaq üçün çox vaxt tələb olunacaq, lakin hazır proqram funksiyaları şimşək sürrətində yerinə yetirəcək və onu (proqramı) yenə və yenə başlatmaq olacaq.

Mənbə kodu (source code)

Yüksək səviyyəli (misal üçün C++) proqramlaşdırma dilində yazılmış proqram. Mənbə kodu C++ dilinin instruksiyalarında ibarətdir, hansı ki elə proqramın əsasını təşkil edir. Maşın dili, əvvəla qeyd etdiyimiz kimi sıfırlardan və birlərdən ibarətdir, lakin adi halda o onaltılıq şəklində verilir, ona görə də maşın kodu aşağıdakına oxşayır:

```
08 A7 C3 9E 58 6C 77 90
```

Bu kodun nə etdiyini bilmək çətindir, elə deyilmi? Siz kommandaların bütün kodlarını tapmayanadək siz bunun nə etdiyini bilməyəcəksiniz – həm də indi çox az adamlar proqram yazmaq üçün maşın dilindən istifadə edir. Bundan başqa, mənbə kodu bir az inglis dilinə oxşayır. Misal üçün, C++ dilinin mənbə kodu belə şəkildədir:

```
if (salary < 0)
    print_error_message();
```

Instruksiya (statement)

Adətən C++ dilində bir sətrdir. Kobud desək, C++ dilində instruksiya şəxsi dilinə oxşayır.

İstifadəçi (user)

Proqramı başladan adam, və yaxud kompüteri nəsə etmək üçün istifadə edən adam istifadəçi adlanır.

C++ dili nə ilə fərqlənir?

Java, Pascal, Fortran, C++ və s. əvvəla qeyd etdiyim kimi yüksək səviyyəli proqramlaşdırma dilləridir (*hight-level languages)*; bu o deməkdi ki, onlar maşın koduna oxşar deyil, onların öz açarlı sözləri var (misal üçün “if” və “while”), hansı ki inglis dilinə oxşayır, uzaqlaşmış olsa da.

Bütün bu dillər eyni bir prinsipi yerinə yetirirsə (maşın dilindən fərqli olaraq proqram yazılmasını asanlaşdırır), onda nəyə görə onlar belə çoxdurlar?

Bu dillərin hamısı ayrı-ayrı məqsədlər üçün yaradılıb. Misal üçün Basic dili, asan öyrənmə və istifadə etmə üçün yaradılıb. Nəticədə o, azad sintaksisi icazə verirdi və pis vərdişlərə öyrəşdirirdi. Buna baxmayaraq Microsoft korporasiyası Visual Basic yaratdı və o Windows əməliyyat sistemi üçün proqram yaratmaq üçün güclü və tez dilə çevrildi.

Pascal dili təhsil müəsissələrində çətin proqramlaşdırmanı öyrənmək üçün yaradılıb. Əgər Basic dili tez, ciddi sintaksis qaydalarına əməl etmirsə, onda Pascal dili isə əksinədir.

C dili əvvəla əməliyyat sistemləri yazmaq üçün yaradılıb. O daha sıx proqramlar yazmaq üçün nəzərdə tutulub. Sonralar C dili proqramçılar arasında daha da populyar olub. C dilinin digər xüsusiyyəti də odur ki, onun az məhdudiyyətləri var, ona görə də maşın dilində lazım olan işləri C – də yazmaq olar.

C++ dili haqqında nə demək olar?

C ilə C++ dilinin əsas fərqləri bundadır ki C++ dili C dilinin davamıdır (təbii ki), həm də o *obyektli proqramlaşırma* dili sayılır. Belə giriş əsasən çətin məsələləri həll etmək üçün nəzərdə tutulub.

Bəs Java və C# ?

1980-ci illərin axırlarında obyektli proqramlaşdırma populyar olub. C dilinin obyektli proqramlaşdırma dilinin versiyasını yaratmaq üçün addım atılıb. Byern Strautskup (Bjarne Stroustrup) birinci buna oxşar dil yaradıb. Bu dil C++ - dur, hansı ki bizim günümüzdə də geniş istifadə olunur.

Lakin C++ dili axırıncı obyekti proqramlaşdırma dili deyil (C-dən gələn). İki yeni dillər – Java və C# dilləri C və C++ dillərinə oxşayır. Lakin bu dillərin hərəsi özünün fərqləri var.

Bu üç dillər arasında bir sıra fərqlər var. C++ dili yaradılırdı bir şərtlə ki, onun çox hissəsi C dilinə oxşayacaq.

Java və C# dilləri proqramlar yazmaq üçün nəzərdə tutulub və onların vasitəsilə əməliyyat sistemlərini yazmaq mümkün deyil. Onların sintaksisləri C/C++ dilinə oxşasalar da, onların bəzi yaddaş ünvanlarına girişi yoxdur. Həm də adamlar bunların yaxşı obyektli proqramlaşdırma dillərinin olduğunu hesab edir.

Java və C# dillərinin sintaksisləri arasında fərq elə də böyük deyil. Java dili Sun Microsystems kompaniyası tərəfindən platformadan asılı olmayan şəklində yaradılıb. C# dili Microsoft korporasiyası tərəfindən yaradılıb, elə onun .NET platforması üçün.

Dediyim kimi, C++ dili əməliyyat sistemləri yazmaq üçündür. Lakin siz onu həm də kommersial proqram və oyunlar proqramı yaratmaq üçün də istifadə edə bilərsiniz. Bu dil daha da azaddır və aşağı səviyyəli səhvləri düzəltməyə imkan verir.

Xoşagələn o xəbərdir ki, C++ dilini bilərək Java və C# dilini öyrənmək sizə olduqca asan olacaq. Həm də C dilində təcrübəsi olan adamlara C++ dilini öyrənmək belə asandır.

C++ dilində proqramın generasiyası

Proqramı yazmaq, əslində prilojeniya yaratmağın birinci addımıdır. Digər bölmələrdə mən bütün keçmək üçün yolları görsədəcəm.

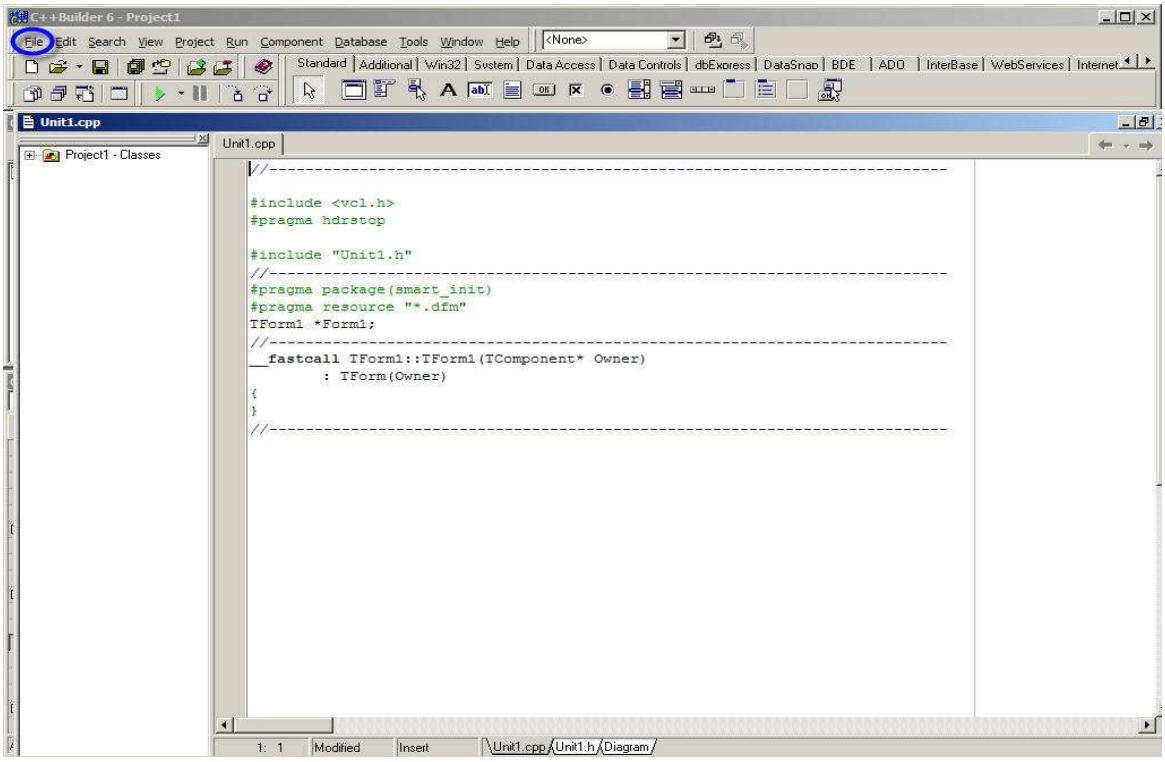
Proqramın instruksiyalarını yazmaq

C++ dilində proqram yazmaq üçün, hər hansı üsul ilə onun instruksiyalarını yazmaq lazımdır. Bunun üçün cüt üsullar mövcuddur:

- ✓ Hər hansı mətn redaktorundan istifadə etmək olar, misal üçün Microsoft Word və yaxud Notepad (ikincisi sistem ilə birlikdədir). Əslində bunu istənilən mətn redaktoru vasitəsilə etmək olar.
- ✓ Mətni həm də istehsal mühitində (İDE – İntegrated Development Environment) yazmaq olar. İstehsal mühiti özündə çoxlu komponentlər, alətlər və s. daşıyır ki bu da proqram yazılmasını olduqca asanlaşdırır. C++ üçün Microsoft kompaniyasından Visual C++ və yaxud Borland kompaniyasından C++ Builder istifadə edə bilərsiniz.

QEYD: Borland C++ Builder istifadə etmək məsləhət görülür. Aşağıdakı çərçivədə onu quraşdırmaq və istifadə etmək qaydaları izah olunur.

Bu proqramı diskdə almaq və yaxud internetdən yükləmək ilə əldə etmək olar. Borland C++ Builder 6 proqramının təxmini ölçüsü 188 MB-dır. Onu əldə edəndən sonra quraşdırırıq, aktiv edirik və başladırıq:



Siz artıq proqram yazmağa və yaratmağa başlaya bilərsiniz. Lakin bizim işimiz yalnız MS-DOS – dur. Ona görə də

File>New>Other>Console Wizard

Proqramın generasiyası

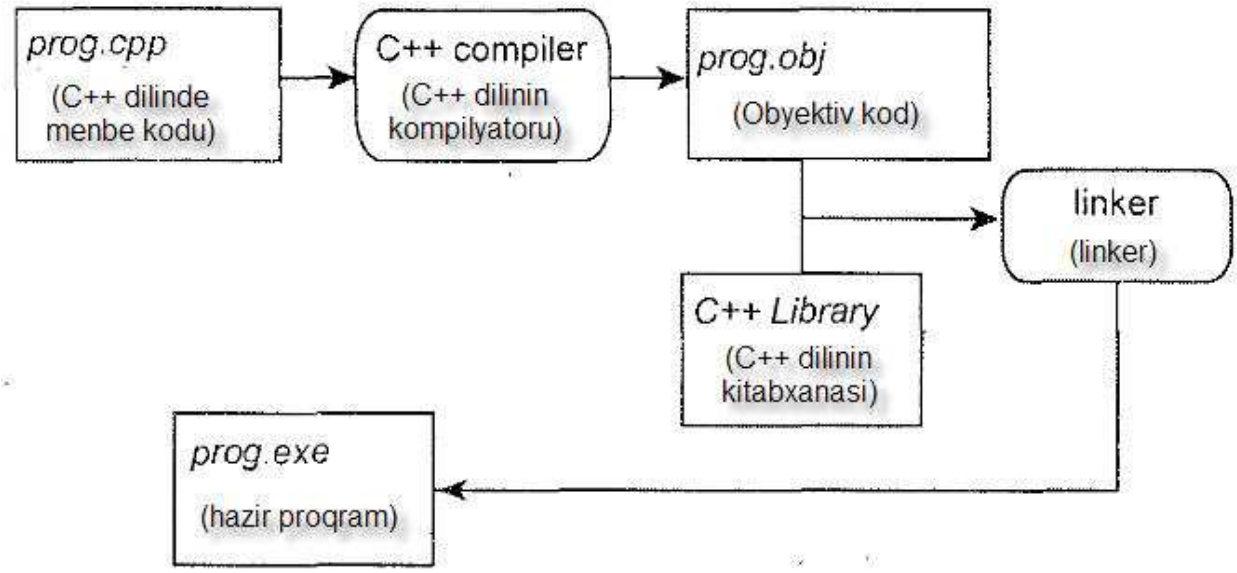
Proqramın generasiyası – bu sizin mənbə kodunun (C++ dilinin instruksiyaları) prijoleniyaya çevrilmə prosesidir. Əgər proqram düzgün yazılıbsa, onda bu proses düymə basmaq kimi asandır.

Birinci mərhələdə proqram *kompilyasiya olunur*; bu deməkdir ki, C++ dilində yazılan mənbə kodu maşın koduna çevrilir (həm də "obyektiv kod" adlanan). Əgər bu mərhələ uğurla başa çatıbsa, onda növbəti mərhələdə proqram-yığıcı başlanır və yaxud *linker*, hansı ki maşın kodunu və C++ dilinin kitabxana kodunu bir yerə yığır.

C++ dilinin kitabxanaları özündə funksiyalar daşıyır, hansı ki, ümumi məsələlərin həlli üçün başlanır. (*Funksiya* – altproqramın digər adı). Misal üçün, kitabxana *sqrt* kvadrat kökü daşıyır, ona görə də sizə kvadrat kökü tapmaq lazım gəlməyəcək. Kitabxanalar həm də altproqramlar daşıyır, hansı ki verilənləri monitora göndərir və sərt diskə faylları necə yazıb-oxumağı bilirlər.

Aşağıdakı şəkildə proqramın generasiyası prosesi görsədilir. Yadda saxlayın ki, əgər siz istehsal mühitinindən (İDE) istifadə edirsinizsə, onda sizin üçün bu proseslər gizli olacaq, siz sadəcə düyməni basacaqsınız.

Əgər proqramın generasiyası uğurla başa çatıbsa, özünüzü təbrik edə bilərsiniz: nə kompilyator, nə linker səhvlər müəyyən etmədi. Bu deməkdir ki, bu sondur? Əlbəttə ki yox. Kompilyator qrammatik (sintaksis) səhvləri tapır. Lakin çoxlu səhvlər var ki, kompilyator onları tapmaq gücündə deyil.



Belə analoqiyaya diqqət yetirək. Tutaq ki, bizdə belə bir cümlə var:

The moon is made green cheese. (Ay yaşıl pendir alınıb).

İnglis dilində belə cümlə qrammatik tərəfdən səhv sayılır. Qrammatikanı düzəltmək üçün “of” sözü əlavə etmək lazımdır:

*The moon is made **of** green cheese.* (Ay yaşıl pendir**dən** alınıb).

İndi cümlədə qrammatik (sintaksis) səhvlər yoxdur. Lakin cümlə qrammatik tərəfdən düz olsa da, bu doğru deyil. Onu doğru etmək üçün “not” sözünü əlavə etmək lazımdır:

*The moon is **not** made of green cheese.* (Ay yaşıl pendirdən düzəlməyib).

Proqramlaşdırma dilləri buna uyğun şəkildə düzəlib. C++ kompilyatoru diqqət yetirir: proqramın sintaksis səhvləri var yoxsa yox, əgər varsa o səhv olan sətri göstərir, əks halda işinə davam edir. Lakin “*proqram hər halda düz işləyir?”* sualına cavab kimi “xeyr” vermək olar. Ona görə də proqramın düz olmasını həm də bu tərəfdən baxmaq gərəkdir.

Proqramın testlənməsi

Proqramı uğurla generasiya edəndən sonra, proqramın hər halda düz işlədiyinə sübut almaq üçün onu bir neçə dəfə başlatmaq lazımdır. Yəni, görmək lazımdır ki, proqram elə siz istədiyiniz işi görür?. Əgər siz tərəfdən yaratdığınız proqramı digər istifadəçilər (sizdən başqa) də istifadə edərlərsə, onda proqramı çox sayda başlatmaq lazımdır. (Əslində böyük kompaniyalarda (proqram təminatı istehsal edən) böyük departmament işləyir, hansı ki, proqramçıların yaratdıqları proqramları testləndirməkdən başqa heç nə ilə məşğul olmurlar).

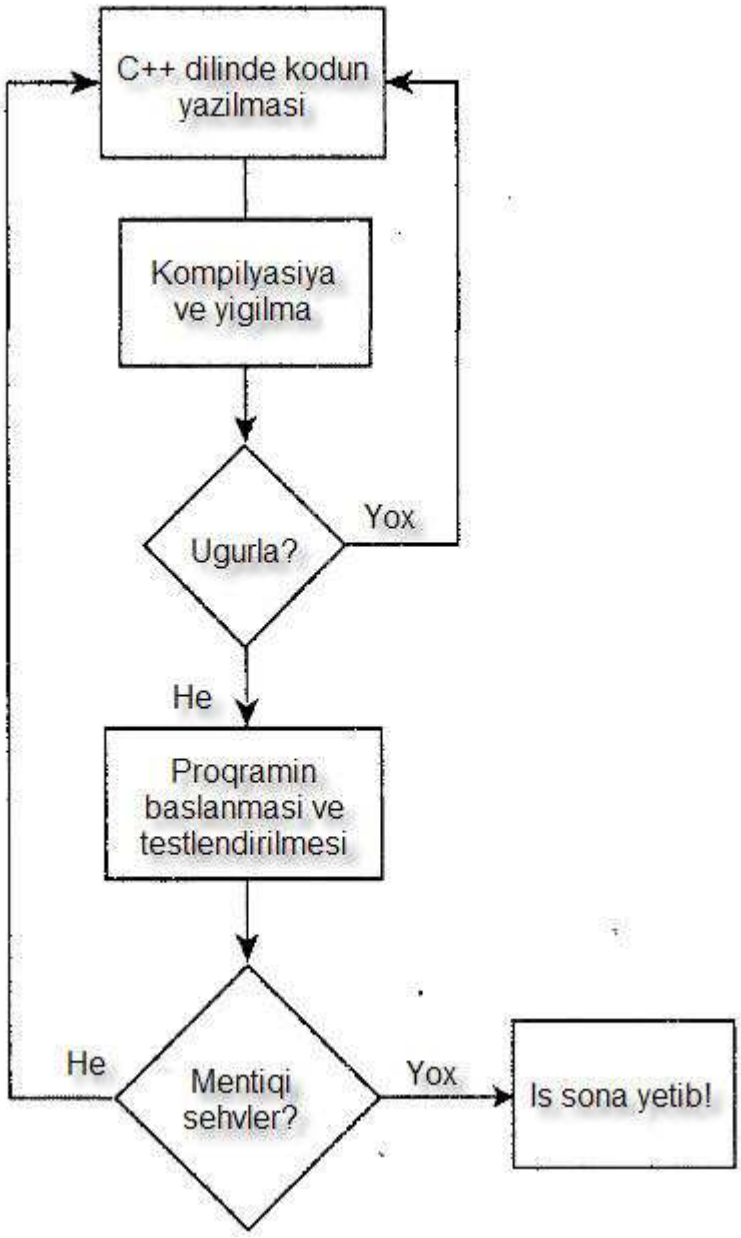
Bu mərhələdə axtardığınız səhvlər *“proqramın məntiqi səhvləri”* adlanır (*proqram-logic errors).* Bu halda proqramda sintaksis səhvlər yoxdur (yəni vergüllər səhv yerdə qoyulmayıbdır və s.), lakin proqram bizim istədiyimiz kimi işləmir.

Proqramın məntiqi səhvləri sintaksis səhvlərinə nisbətən daha çətin tapılan ola bilər. Tutaq ki, proqram səhv rəqəmi verir və yaxud o öz işini görünməyən səbəbdən bitirir. Hansı instruksiyada səhvə yol verilib? Suala cavab tapmaq həmişə asan olmur. Proqramın testləndirilməsi və problemin (-lərin) müəyyən olunması prosesi *qaydalanma (debugging)* adlanır.

Lazımi şəkildə düzəltdirmə

Əgər proqram öz işini düz görürsə, onda iş sona yetib. Lakin əgər proqram yuxarıda danışdığımız kimi məntiqi səhvlər müəyyən olunubsa, onda onları aradan qaldırmaq vacibdir.

İş belə şəkildə gedir:



Misal 1.1. İsmarıcın çapı

Proqramlaşdırmağa başlamaq üçün İDE Borland C++ Builder proqramını başladın və

File>New>Other>Console Wizard

Açılan pəncərədə bu kod olacaq:

Kod 1.0----------

```
#pragma hdrstop
#pragma argsused
int main(int argc, char* argv[])
{
      return 0;
}
```

Bütün kodu təmizləyib və bunu yazırıq (və yaxud pozmadan , yalnız lazım gələnləri dəyişdiririk):

Kod 1.1----------

```
#include <iostream>
using namespace std;
int main()
{
      return 0;
}
```

Indi isə, birinci misalı göstərmək üçün **cout** operatorunu yazırıq:

cout << “Men C++ dilini oyrenirem!!!”;

Yəni, əvvəlki kodu pozmuruq, sadəcə olaraq yuxarıdakı sətri “**{“** və ” **}”** işarələri arasında yazırıq. Belə şəkildə:

Kod 1.2----------

```
#include <iostream>
```

```
using namespace std;
int main()
{
        cout << “Men C++ dilini oyrenirem!!!”;
        return 0;
}
```

QEYD: Arasında kod olan uzun xətləri yazmaq lazım deyil. Onları mən sadəcə olaraq kodu qeyd etmək üçün yazmışam!

Kod 1.2 – də olan kodda yeddinci sətrdə yazdığımız cout – da çoxlu probellər qoymaq lazım deyil. Siz həm də bunu “Tab” düyməsini basmaqla edə bilərsiniz. Yaxud heç etməyə də bilərsiniz. Mən onu rahatlıq üçün etmişəm. Kod 1.2 və Kod 1.3 arasında heç bir fərq yoxdur:

Kod 1.3---

```
#include <iostream>
using namespace std;
int main()
{
cout << “Men C++ dilini oyrenirem!!!”;
return 0;
}
```

Lakin siz bir neçə şeyə diqqət yetirməlisiniz. C++ dilində böyük və kiçik hərflər arasında fərq var. Ona görə də əgər siz kodun heç olmasa bir hərfini böyük hərflə yazsanız, kompilyator bunu səhv kimi qəbul edəcək. Misal üçün:

Kod 1.4---

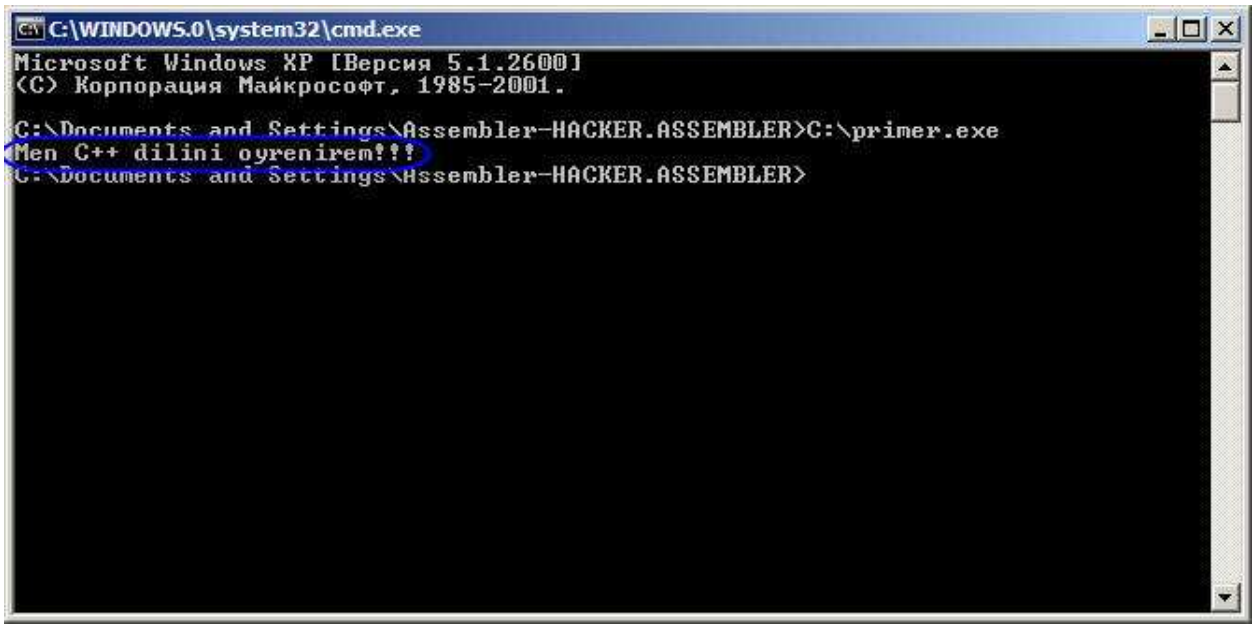
```
#include <iostream>
using namespace std;
int main()
{
        Cout << “Men C++ dilini oyrenirem!!!”;
        RETURN 0;
}
```

Bu səhvdir! Siz indi yəqin fikirləşirsiniz: “Mən” sözü də böyük hərflə başlanır axı, bəs o səhv sayılmır? Xeyr, sayılmır! Çünki bu söz ekranda görsədiləcək. Lakin kod ekranda görsədilmir, ona görə də onu biz istədiyimiz kimi yaza bilərik.
Bir də ki, hər instruksiyanın sonunda “**;**” işarəsini yazmağa yaddan çıxartmayın.

Belə, kodu yazandan sonra proekti yadda saxlayıb (bütün faylları eyni bir qovluğa!) proqramı başladın. Qara pəncərə (MS-DOS pəncərəsini) görsədən kimi proqram öz işini sona yetirəcək. Əlbəttə ki, biz “Mən C++ dilini öyrənirəm!!!” sətrini görə bilməyəcəyik. Əslində o görsənir, lakin biz onu görmürük. Görmək üçün onu MS-DOS sətrində başlatmaq lazımdır. Necə? Misal üçün hazır yaratdığımız proqram “C:\” bölməsində yerləşir. Onda, **RUN>cmd** yazmaqla, və yaxud C:\Windows\system32\cmd.exe proqramını başlatmaqla MS-DOS sətrini başladırıq, sonra orada yazırıq:

C:\”yaratdigimiz proqramin adi”.exe

və ENTER düyməsini basırıq. Və buna oxsar belə bir şey görürük:



Bu halda, mən yaratdığım proqramın adını “primer” qoymuşam. Belə, biz istədiyimizə nail olduq, yəni biz MS-DOS sətrlərində “Mən C++ dilini oyrenirem!!!” cümləsini gördük.

Bu necə işləyir

İstəyirsinizsə - inanın, istəmirsinizsə - inanmayın, lakin bu proqram yeganə bir əsl instruksiyadan ibarətdir. Bu halda, digər mətni şablon hesab edə bilərsiniz, hansı ki, yazmaq elə də vacib deyil, lakin gərəkdir!

Aşağıda görsətdiyim sintaksisdə standard elementlər qalın şriftlə qeyd edilib. Onların nə üçün lazım olduğuna narahat olmayıb, sadəcə istifadə edin (yəni, onları pozmayın):

Kod 1.5----------

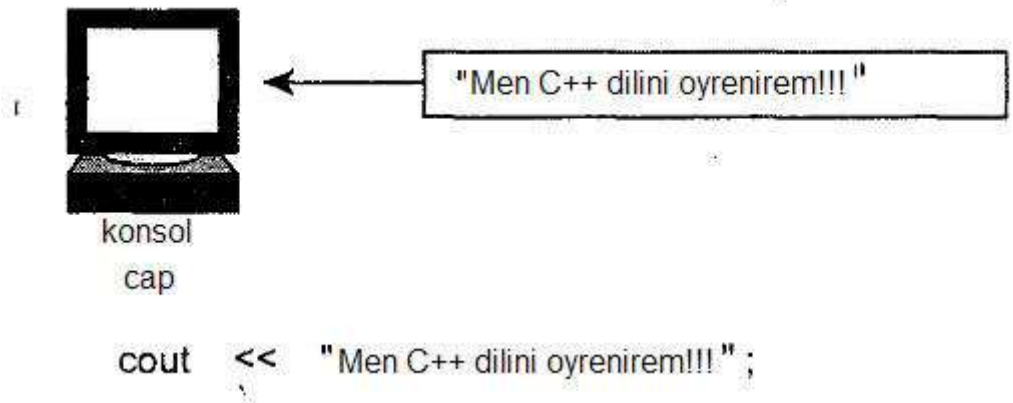
```
#include <iostream>
using namespace std;
int main()
{
    Burada sizin instruksiyalarınız !
    return 0;
}
```

Bu proqramda yeganə bir faktiki instruksiya var (hansı ki, biz onu yeddinci sətrə yazırıq):

```
cout << “Men C++ dilini oyrenirem!!!”;
```

cout nədir? Bu – *obyekt*, konsepsiyadır, hansı ki mən onu kitabın ikinci başlığında aydın izah edəcəm. cout operatoru MS-DOS sətrlərinə yazdığımız sözləri görsədir. Siz ekrana nəsə göndərən zaman, o nəzərdə tutulan şəkildə görsənəcək.

C++ dilində siz sözləri **cout** obyekti ilə və sol tərəfə yönələn “<<” operatoru ilə çap edirsiniz. Əgər aşağıda görsədilən şəkildə siz onu vizual kimi anlasanız, siz cout obyektini heç vaxt səhv istifadə edə bilməyəcəksiniz:



Nöqtə ilə vergülləri yaddan çıxartmayın. Onlar hər instruksiyanın sonunda yazılmalıdır, bir neçə hallar istisna olmaqla.

Texniki səbəblərə görə cout obyekti həmişə sol tərəfdə olmalıdır. Belə şəkildə bütün verilənlər sol tərəfə yönəlir.

Aşağıdakı cədvəldə cout obyektindən istifadə etməyin digər bir neçə nümunələri var:

İnstruksiya (statement)	Hereket (action)
cout << “RASIM YAXSI PROQRAMCIDIR” ;	“RASIM YAXSI PROQRAMCIDIR” mətnini çap edir
cout << “C++ cox gozel dildir!”;	“C++ cox gozel dildir!” mətnini çap edir
cout <<” 25 + 2 = 27”;	“25 + 2 = 27” mətnini çap edir

Məsələlər

Məsələ 1.1.1. *Proqram yazın, hansı ki, “Men proqramci olmaq isteyirem” cümləsini çap etsin. Əgər siz elə əvvəlki kodda yazmaq istəyirsinizsə, onda edin. (Kömək: Bütün mətni saxlayın, yalnız “Men C++ dilini oyrenirem!!!” cümləsinin yerinə “Men proqramci olmaq isteyirem” cümləsini yazın).*

Məsələ 1.1.2. *Proqram yazın, hansı ki sizin tam adınızı ekrana çap etsin.*

#include direktivası və “using” instruksiyası haqqında nə?

Mən demişdim ki, proqramın beşinci sətri faktiki instruksiya sayılır. Mən birinci sətri buraxmışam.

#include <iostream>

Bu misal C++ dilinin preprosessor direktivasdır (*preprocessor directive).* Direktiv şəklində:

#include <filename>

bəyannamələr və qeydlər yükləyir, hansı ki, C++ dilinin standard kitabxanasında mövcuddur. Bu direktiv olmadan siz cout obyektini istifadə edə bilməzsiniz. Əgər siz C və C++ dillərinin köhnə versiyalarınında istifadə etmisinizsə, siz maraqlana bilərsiniz ki, nəyə görə müəyyən fayl elan olunmur (*.h fayl). iostream faylı əlavə olunan virtual fayl sayılır, hansl özündə informasiya daşıyır.

Əgər siz C++ dilində başlayansınızsa, onda sadəcə yadda saxlayın ki **#include <iostream>** direktivasını həmişə əlavə etmək (yazmaq) lazımdır. Sonralar, riyazi məsələləri həll etmək üçün, misal üçün sqrt kvadrat kökündən istifadə etmək üçün
#include <math.h> direktivasını əlavə etmək (yazmaq) lazım olacaq.
Bu əlavə iş hesab olunur? Bəli. C++ dili bunlarsız yaradıla bilərdi? Ola bilər. C/C++ dillərində professional proqramçılar kitabxanalardan qaçırlar və yaxud onları modifikasiya edirlər.

Sizə həm də **using** instruksiyasını yazmaq lazımdır. Bunula siz obyektlərə, misal üçün std::cout, birbaşa ünvanlana bilərsiniz. Bu instruksiyanı (**using namespace std;)** yazmasaz, onda sizə ismarıcı belə şəkildə yazmaq lazım gələcək:

std::cout << “Men C++ dilini oyrenirem!”;

Digər sətrə keçid

C++ dilində yazılan mətn avtomatik olaraq növbəti fiziki sətrə keçmir. Keçid almaq üçün *növbəti sətrə keçid simvolu* (newline) yazmaq lazımdır. Əgər siz sətrə keçid simvolunu yazmasınız, onda bütün sözlər eyni sətirdə yerləşəcək. (İstisna hal: bir sətrdə olan sözlər sətrin axırına çatan kimi avtomatik olaraq növbəti sətrə keçid alır).

Növbəti sətrə keçid almaq üsullarından biri **endl** konstantını yazmaq lazımdır:

std::cout <<”Men C++ dilini oyrenirem!!!” << std::endl;

Lakin əgər siz

using namespace std;

sətrini əlavə etmisinizsə, onda siz rahatlıq üçün belə yaza bilərsiniz:

cout << “Men C++ dilini oyrenirem!!!” << endl;

“endl” sözü “end line” sözlərinin qısalmış formasıdır, hansı ki tərcümədə “sətrin sonu” mənasını verir.

Digər sətrə keçmək üsullarından biri də **\n** əlavə etməkdir. Misal üçün:

cout << “Men C++ dilini oyrenirem!!! \n”;

Misal 1.2. Bir neçə sətrin çapı

Bu bölmədə proqram bir neçə sətirdə sözləri çap edir.

Kod 1.4----------

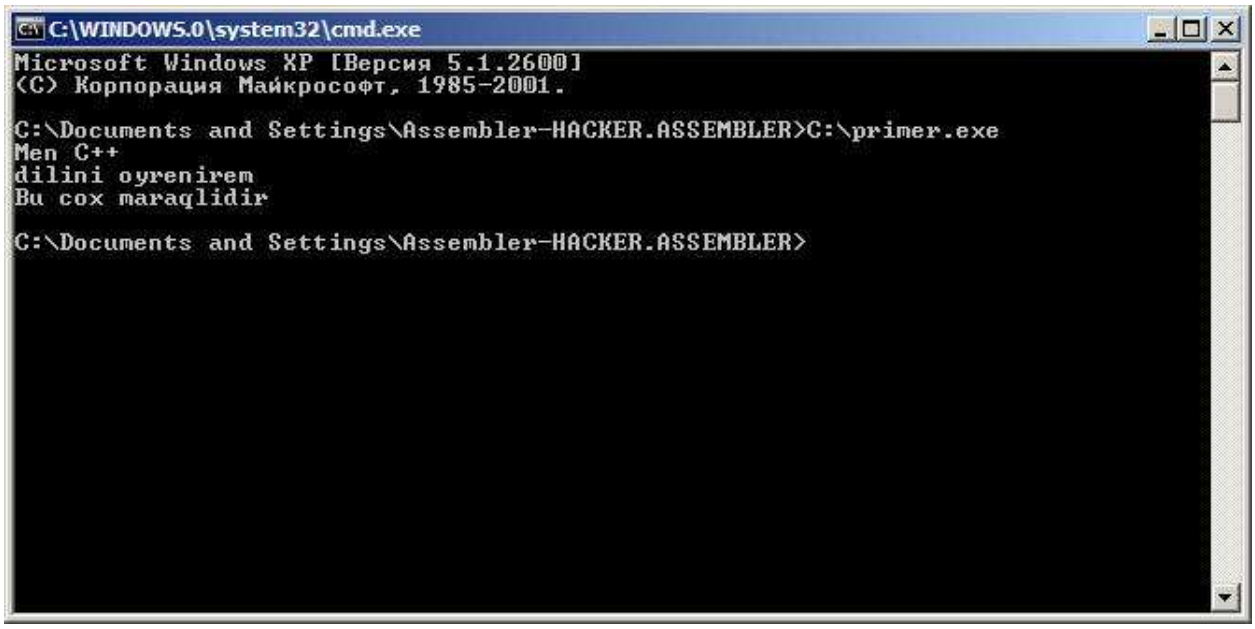
```
#include <iostream>
using namespace std;
int main()
{
	cout << “Men C++”<< endl;
	cout << “dilini oyrenirem”<< endl;
	cout << “Bu cox maraqlidir” << endl;
return 0;
}
```

Kodu yazın, proekti **print2** adında saxlayın və ondan sonra kompilyasiya edin və başladın.

Əgər siz hər şeyi düz etmisinizsə, onda proqram belə şəkildə sözlər çap edəcək:

```
Men C++
dilini oyrenirem
Bu cox maraqlidir
```

MS-DOS – da isə:



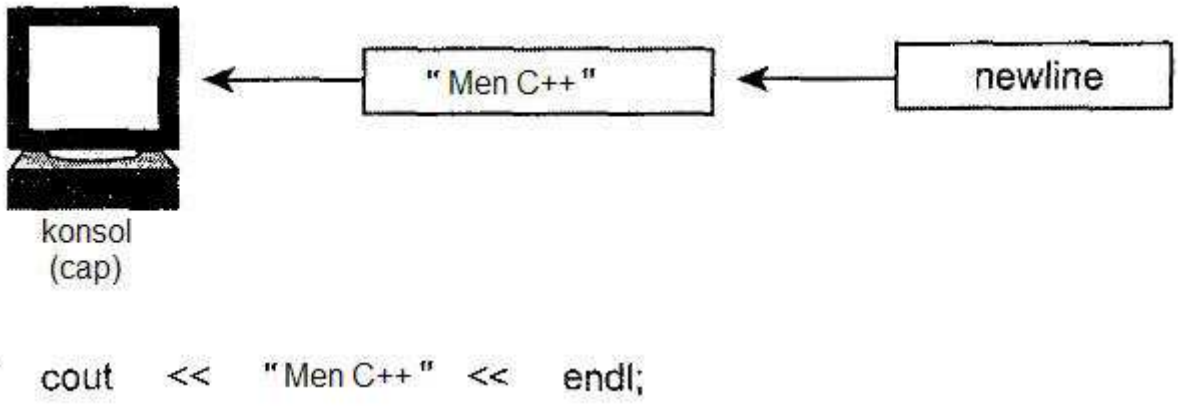
Bu necə işləyir

Bu nümunə mənim görsətdiyim birinci nümunəyə bənzəyir. Əsas fərq bundadır ki, bu nümunədə növbəti sətrə keçiddən istifadə olunub. Əgər bu simvolları yazmasaydıq, onda belə olacaqdı:

Men C++ dilinioyreniemBu cox maraqlidir

Lakin bu bizə lazım deyil.

Aşağıdakı şəkil, ikinci proqramın işini göstərir:



Siz konsola bir instruksiya ilə istənilən qədər sözlər göndərə bilərsiniz:

cout << “Menim adim Rasimdir.” << “Men bu kitabi sizin ucun yazmisam.”;

Onlar eyni sətrdə olacaqlar:

Menim adim Rasimdir.Men bu kitabi sizin ucun yazmisam.

Siz həm də sətrə keçəni bir instruksiyada istifadə edə bilərsiniz:

cout << “Menim adim” << endl << “Rasimdir. “;

Hansı ki, konsolda belə bir şey verəcək:

Menim adim
Rasimdir.

Məsələlər

Məsələ 1.2.1. *Bir instruksiyadan istifadə etməklə aşağıdakı sözləri çap edin:*

Assembler dili - maşın dilidir.

Məsələ 1.2.2. *Elə proqram yazın ki, o hər cümlə arasında boş sətir saxlasın. Cümlələri özünüz seçin.*

Sətir nədir?

Ən əvvəldən mən mətn istifadə etmişdim, hansı ki işarələr arasında durur:

```
cout << “Men C++ dilini oyrenirem!!!”;
```

İşarələrdən xaricdə yerləşən hamısı dilin sintaksisidir, işarələr arasında yerləşən isə verilənlərdir.
Faktiki olaraq, kompüterdə yerləşən bütün verilənlər rəqəmsaldır. Onların necə istifadə olunmasından asılı olaraq, onlar sətrlər ola bilər, hansı ki, çap olunan sözlərdən ibarətdir. Bu doğrudur.

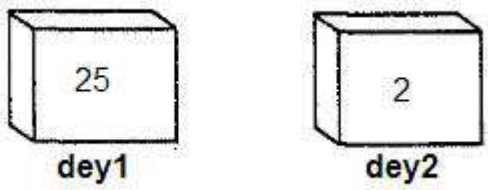
Yəqin siz haradansa ASCII kodu haqqında eşitmisiniz. “Men C++ dilini oyrenirem” – bu nümunədə ASCII verilənləridir. “M”, “e” , “n” və s. simvolları isə hər baytda yerləşir, hansı ki, hərəsinin rəqəmli kodu olur.

Mən bu haqqında daha aydın 7-ci başlıqda danışacam. Yadda saxlamaq vacibdir ki, “” işarələri arasında yerləşən verilənlərdir. Verilənlərin bu növü sətir adlanır.

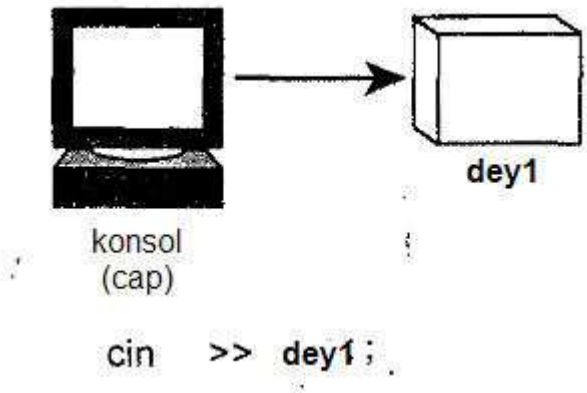
Verilənlərin saxlanması: C++ dilinin dəyişənləri

Əgər C++ dilində ancaq gic-gic ismarıcları çap etmək olsaydı, onda bu dil belə faydalı olmazdı. Məqsəd adətən belə olur – hər hansı yerdən yeni verilənləri almaq və onlar üzərində hər hansı iş aparmaq.

Belə əməliyyatlar dəyişənləri tələb edir (*variables):* onlar yeşiklərə oxşayırlar (lakin gözə görsənmirlər), hansı ki, onlara nəsə yerləşdirmək olar. Proqramın işləyən zamanı bu yeşiklərə (dəyişənlərə) nəsə yazmaq, oxutmaq olar. Növbəti nümunədə iki - **dey1** və **dey2** dəyişənləri görsədilib:



Dəyər dəyişənə necə yerləşdirilir? Üsullardan biri də konsoldandır. C++ dilində dəyərləri yazmaq üçün **cin** obyektindən istifadə edə bilərsiniz. cin obyekti ilə >> operatoru istifadə olunur.



Bu instruksiyaya cavab kimi aşağıdakılar baş verir:

- ✓ Proqram dayanır və istifadəçinin nəsə yığdığını gözləyir.
- ✓ İstifadəçi rəqəm yazır və ENTER düyməsinə basır.
- ✓ Rəqəm qəbul olunur və **dey1** dəyişəninə yazılır (hazırki halda).
- ✓ Proqram öz işini bitirir.

Bunun hamısını kod ilə yazsaq

cin >> dey1;

Lakin C++ dilində dəyişəni istifadə etməzdən əvvəl, onu elan etmək lazımdır. Bu C++ dilinin ciddi qaydasıdır və elə bu onu Basic dilindən fərqləndirir, hansı ki o dildə dəyişənləri elan etmək lazım gəlmir.

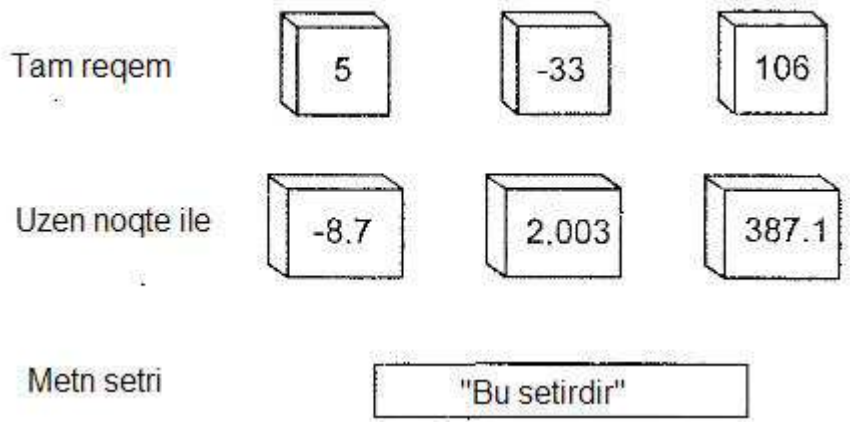
Bu vacibdir, ona görə də mən bunu bir qayda kimi edirəm:

- ■ **C++ dilində dəyişənləri istifadə etməzdən qabaq onları elan etmək lazlmdır.**

Verilənlər növlərinə giriş

Siz dəyişənləri sehirli yeşiklər kimi təsəvvür edə bilərsiniz, hansı ki onlara informasiya - daha doğrusu *verilənlər* yaza bilərsiniz. Bəs hansı verilənlər növü?

Kompüterdən olan bütün verilənlər rəqəmsaldır, lakin onlar üç əsas formatda təşkil olunub: tam rəqəm (*nteger)*, üzən nöqtə ilə (*floating-point)* və mətn sətri.



Tam rəqəm və üzən nöqtə formatları arasında fərqlər çoxdur, lakin əsas qayda budur:

- **Əgər misal üçün 2.1 (iki tam onda bir) və s. növündə olan verilənləri yadda saxlamaq lazımdırsa, onda üzən nöqtə növünü seçin, əks halda tam rəqəmdən istifadə edin.**

Üzən nöqtə ilə olan əsas verilənlər növü C++ dilində **double** – dır. Ad sizə qəribə görsənə bilər: “double-precision floatin point” (ikilik dəqiqliyində üzən nöqtə). Dildə həm də tək dəqiqli üzən nöqtə növü var **float**, lakin o çox az işlənir. Az səhvlərin olması üçün double-dən istifadə edin.

double verilənin elan edilməsi sintaksisi belədir:

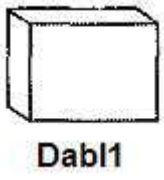
double verilənin_adı;

Onu həm də bir çox verilənlərin elan edilməsi üçün də istifadə edə bilərsiniz:

double verilənin_adı1, verilənin_adı2, verilənin_adı3, ...;

Misal üçün, aşağıdakı instruksiya **Dabl1** adında dəyişəni elan edir:

double Dabl1;



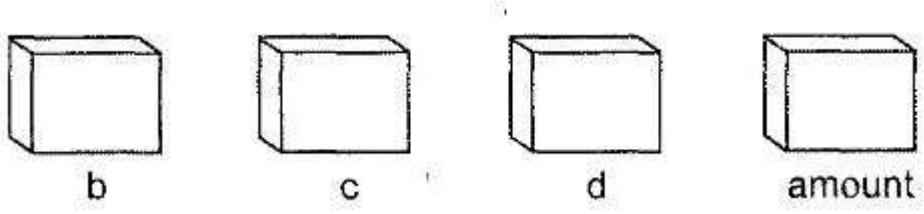
Növbəti instruksiya daha çətin sintaksis daşıyır, yəni dörd eyni növdə dəyişən elan edir: **b, c, d** və **amount** adında.

double b, c, d, amount;

Həm də bunu aşağıdakı şəkildə yazmaq olar:

double b;
double c;
double d;
double amount;

Bu instruksiyaların nəticəsi, dörd eyni növdə dəyişən elan edilməsidir:



Misal 1.3. Temperaturun çevrilməsi

Bir misal olaraq, faranheyt və selsi temperaturlarını çevirək. Bu da çevrilmə düsturu:

Fahrenhelt = (Cilsius * 1.8) + 32

Bunu etmək üçün yeni imkanlardan istifadə etmək lazımdır:

- ✓ Istifadəçinin yığmağını təşkil etmək;
- ✓ Yığılmış verilənləri dəyişəndə yadda saxlamaq;

Aşağıda belə proqram tam yazılıb. Yeni proekt yaradın və onu **convert** adlandırın. Bundan sonra proqramı kompilyasiya edin və başladın.

Kod 1.5----------

```
#include <iostream>
using namespace std;
int main()
{
	double  ctemp, ftemp;
	cout << “Selsi yazin ve ENTER basin:”;
	cin >> ctemp;
	ftemp = (ctemp * 1.8) + 32;
	cout << “Fahrenheit temp is: “ << ftemp;
return 0;
}
```

Proqramların kodlarını oxumaq asan olur, əgər onların (kodun) tərkibində “şərhlər” mövcuddur. C++ dilində şərhləri // işarələrini yazandan sonra yazmaq olar və sətrin axırınıa kimi olan hər şey şərh kimi qəbul oluncaq. Misal üçün:

Kod 1.6----------

```
#include <iostream>
using namespace std;
int main()
{
	double  ctemp, ftemp;  // Burada iki dəyişən elan olunur
	cout << “Selsi yazin ve ENTER basin:”;
	cin >> ctemp;  // Burada istifadəçi verilənləri yazır və yadda saxlanılır
	ftemp = (ctemp * 1.8) + 32;  // Sonra faranheyt tapılır
	cout << “Cavab: “ << ftemp;   // Axırıncı mərhələdə isə cavab ekrana çıxır
return 0;
}
```

Adama kodları şərhlərlə oxumaq olduqca asan olsa da, onları (şərhləri) yazmaq üçün çox vaxt tələb olunur. Bir qayda kimi yadda saxlayın:

- **C++ dilinin kodunda // işarələrindən sətrin axırına kimi gələn sözlər şərhlər hesab olunur və kompilyator tərəfindən rədd edilir. Yəni proqramın prinsipinə heç bir maneçilik törətmir.**

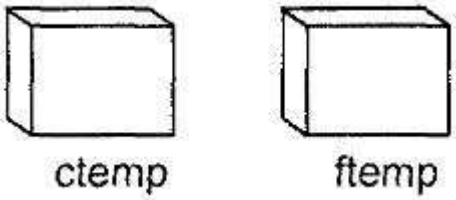
Şərhləri əlavə etmək vacib deyil, lakin gələcəkdə sizə əvvəl yazdığınız proqramın kodunu rahat oxumaq üçün müəyyən yerlərdə onları yazmaq məsləhət görülür.

Bu necə işləyir

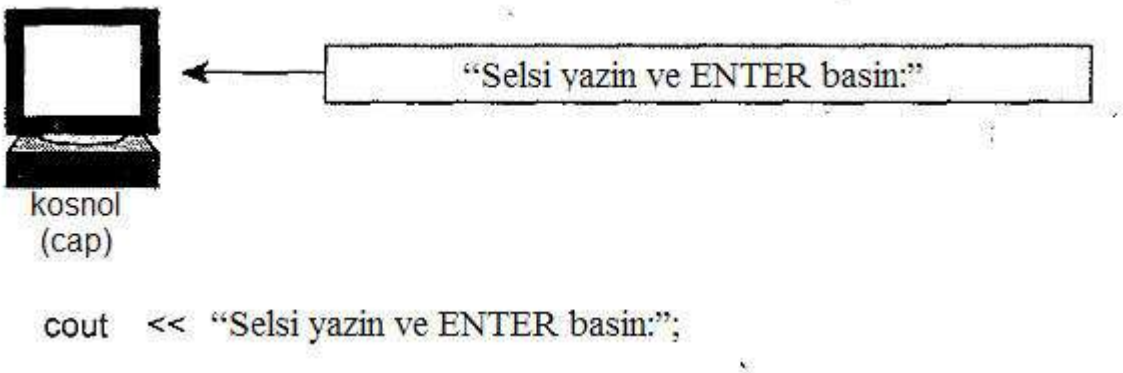
Birinci main instruksiyası iki double növündə dəyişən elan edir. **ctemp** və **ftemp** adlarında.

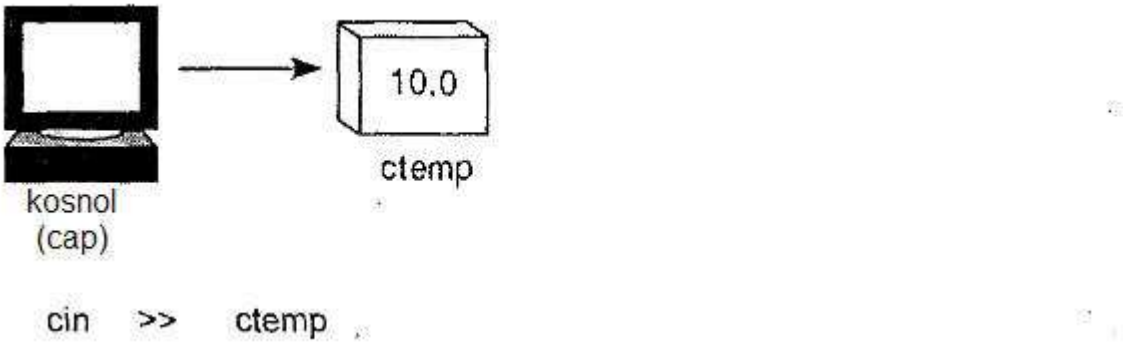
double ctemp, ftemp;

Bu bizə iki yeşik verir, hansı ki onlarda dəyərlər saxlamaq olar.



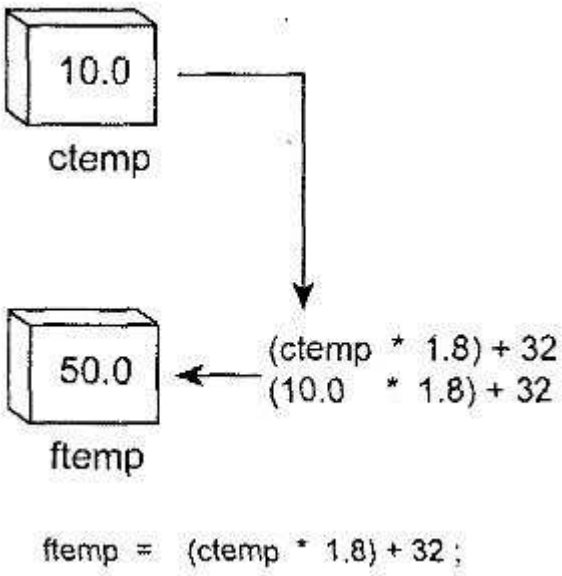
Növbəti iki instruksiya istifadəçi üçü kömək çap edir və istifadəçi yığdığı verilənləri **ctemp** dəyişənində saxlayır. Tutaq ki, istifadəçi 10 yığıb. Bundan sonra 10.0 rəqəmi **ctemp** dəyişənin yadda saxlanılır.



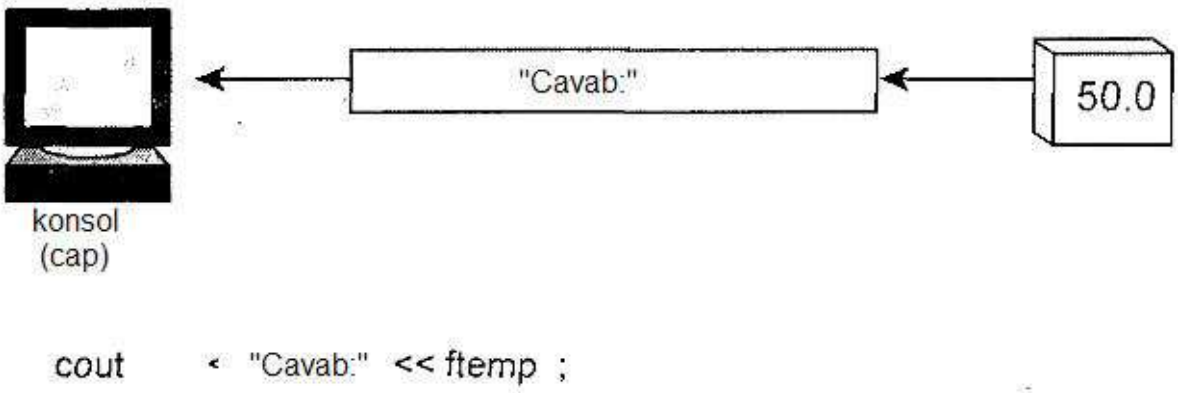


Növbəti instruksiya faktiki çevrilmə edir, hansı ki sonra cavabı **ftemp** dəyişənində yadda saxlanılır:

```
ftemp = (ctemp * 1.8) + 32;
```



Axırıncı mərhələdə proqram nəticəni ekrana göstərir – bizim halımızda 50-dir.



Məsələlər

Məsələ 1.3.1. *Nümunəni elə köçürün ki o əks iş görsün: istifadəçi tərəfindən yığdığı verilənləri* ***ftemp*** *dəyişənində yadda saxlasın, nəticəni isə* ***ctemp*** *dəyişənində saxlasın. Bundan sonra nəticəni çap edin. (Kömək: Əks çevrilmə üçün düstur:* ***ctemp = (ftemp- 32) / 1.8)***

Məsələ 1.3.2. *Cəmi bir* ***ftemp*** *dəyişənindən istifadə etməklə faranhetytdən selsiyə çevrilməsini yazın. Bu 1.3.1 proqramının modifikasiyasıdır.*

Məsələ 1.3.3. *Proqram yazın, hansı ki istifadəçinin yığdığı verilənləri* ***n*** *dəyişənində saxlasın və bu verilənin kubunu tapsın (****n * n *n)****. Əmin olun, instruksiyanın sözü* ***cube*** *sözündən istifadə edir.*

Məsələ 1.3.4. square.cpp *nümunəsini köçürün və orada bütün* ***n*** *adında dəyişənləri* ***num*** *ilə adını dəyişdirin.*

Dəyişənin adları və açarlı sözlər haqqında bir neçə söz

Bu bölmədə **ctemp** və **ftemp** dəyişənləri istifadə olunmuşdur. **ctemp** və **ftemp** adları yerində mən **“terminator2003”, “killerRobot”, “GovernOfCalifornia”** və s. adlarını da qoya biləcəkdim.

Belə, hansı adları yazmaq olar, hansılarını isə yox?

- ✓ Birinci simvol hərf olmalıdır. Birinci simvol rəqəm ola bilməz. Texniki tərəfdən birinci simvol “_” işarəsi də ola bilər, lakin onu istifadə etmək məsləhət görülmür (adın əvvəlində).
- ✓ Adın qalan hissəsi həm hərflər, rəqəmlər, həm “_” işarəsini daşıya bilər.
- ✓ Probellər olmaz!
- ✓ Artıq əvvəlcədən haradasa istifadə olunmuş (kodda) adlarını yenə yazmaq məsləhət görülmür.

C++ dilində hər hansı mənası olan sözlər “açarlı sözlər” adlanır (keywords). Belə sözlərdən biri də *main* sözüdür. Digər sözlər özündə verilənlərin növünü mənasını daşıyır – *int, float, double* və s. Həm də açarlı sözlər – *if, while, do, switch* və *class.*

Birinci başlıqda olanların əsasları

- ✓ Proqramın yaradılması C++ dilində kodun yazılması ilə başlanır. Kod C++ dilinin instruksiyalarından ibarətdir, hansı ki bir az inglis dilinə bənzəyir. Proqram başlanmazdan əvvəl maşın koduna çevrilməlidir, hansı ki, onu kompüter (prosessor) başa düşür.

- ✓ Yuxarı səviyyəli proqramlaşdırma dillərinin kodlarının maşın koduna çevrilməsi prosesinə *kompilyasiya* adlanır.
- ✓ Kompilyasiyadan sonra o, C++ dilinin standard kitabxana funksiyaları ilə yığılır. Bu proses *komponovka (yığılma)* adlanır.
- ✓ Xoşbəxtlikdən sizdə istehsal mühiti var, hansı ki bu proseslər bir funksional düyməni basmaqla yerinə yetirilir.
- ✓ C++ dilində olan sadə proqramların kodu aşağıdakı şəkildədir:

```
#include <iostream>
using namespace std;
int main()
{
        Burada instruksiyalarınız!
return 0
}
```

- ✓ Nəsə çap etmək üçün *cout* obyektindən istifadə edin. Misal üçün:

```
cout << “Men C++ dilini oyrenirem!!!” ;
```

- ✓ Nəsə çap edəndən sonra digər sətrə keçmək üçün *cout* obyektini və sonra *endl* sətrə keçməni istifadə edin. Misal üçün:

```
cout << “Men C++ dilini oyrenirem!!!” << endl;
```

- ✓ C++ dilində demək olar ki hər instruksiya “;” işarəsi ilə bitir.
- ✓ // işarələrindən sonra şərhlər yazmaq olar. Bu kodun oxunmasını asanlaşdırır və kompilyator tərəfindən rədd olunur, yəni proqramın işinə heç bir maneçilik törətmir.
- ✓ Dəyişənləri istifadə etməzdən əvvəl onları elan etmək lazımdır. Misal üçün:

```
double x;   // x adında dəyişən elan olundu, hansı ki onun növü üzən
// nöqtədir.
```

- ✓ İstifadəçinin klaviaturada yığıdığı simvolları (hərfləri, rəqəmləri və s.) hər hansı dəyişənə yadda saxlamaq üçün *cin* obyektindən istifadə olunur. Misal üçün:

```
cin >> x;
```

- ✓ Dəyişənə verilənləri yadda saxlamaq həm də “=” operatoru ilə olar. Misal üçün:

```
x = y * 5; // burada y dəyişəni 5 rəqəminə vurulur və cavabı x
 // dəyişəninə yazılır.
```

BAŞLIQ 2

Həll, həll.

İndi, nə vaxt ki, siz verilənlərin yazıb oxunmasını, həm də onların üzərində işlər aparılmasını bilirsiniz, siz C++ dilində əsl proqramlar yazmaq yolundasınız. Lakin proqramlaşdırma daha ciddi işlər üçün də istifadə olunur (əsasən).

Əslində kompüterlər həllər qəbul edirlər. Bütün bu başlıq elə bu xüsusiyyətə həsr olunub.

Lakin əvvəl verilənlərin növü haqqında bir neçə söz.

Kompüterlərin necə həllər qəbul edildiyini bilməzdən əvvəl, kompüterdə olan verilənlərin necə quraşdırılıdığını bilmək lazımdır. Axırıncı mərhələdə kompüterdə olan bütün verilənlər sıfırlar və birlərdən ibarətdir. Elə bundan *verilənlərin növləri (data types)* yaranıb.

Riyaziyyatı öyrənən zaman sizə verilənlərin növləri haqqında narahat olmaq lazım deyildi. Rəqəm elə rəqəmdir, hansı sonra elə rəqəm olur. Bütün bu riyazi ifadələr ekvivalentdir:

3 3.0 üç 2+1

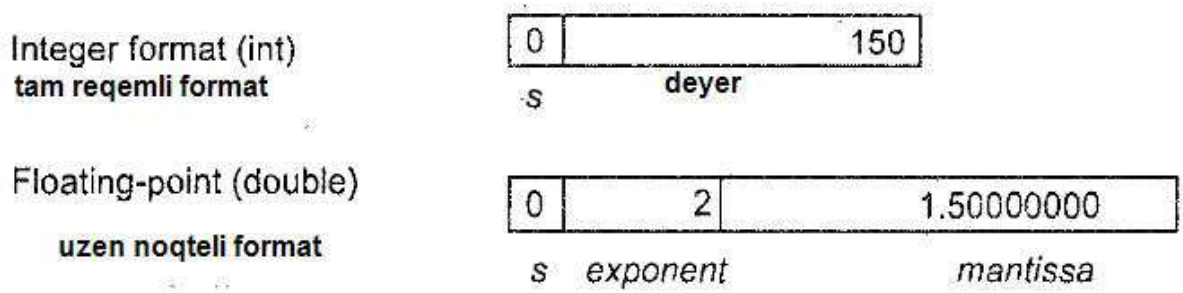
Lakin komüpter dilləri təmiz riyaziyyatdan fərqlənir. Yalnız biliyə malik olmaq bəs deyil, onların yadda saxlanılması üsulunu bilmək vacibdir.

Təmiz riyaziyyat dünyasında fərqli olaraq, kompüter dünyasından olan bütün verilənlər qiymətlidir.

Birinci başlıqda üzən nöqtəli verilənlər növünün nümunələrini yazmışdım. Əsas fərq bundadır ki, üzən nöqtəli növündə rəqəmlərin həm də onluq, yüzlük, minlik və s. hissələri də ola bilər (misal üçün, 2.5 , 5.85 , 10.464 və s.). Lakin tam rəqəm növü bunu bacarmır.

Lakin fərq bununla bitmir. Əgər dərinə baxsaq, tam rəqəmli ilə üzən nöqtəli növləri bir-birinə heç oxşamır.

Aşağıda bir ayrı rəqəm kimi 150 rəqəmi həm üzən nöqtəli, həm də tam rəqəmli növlərində görsədilib.



s işarəli bir ona görsədir ki, rəqəm müsbət yaxud mənfidir; **0** rəqəmi mənfi olmayan rəqəmə göstərir.

Eksponentdən sonra – bu odur ki, tam rəqəmli növünü üzən nöqtəli növündən ayırır. İndi isə 10 rəqəminin 18 dərəcəsinin yadda saxlamaq probleminə baxaq. Bu rəqəm belə yazılır:

1 000 000 000 000 000 000

Bu dəyəri tam rəqəmli növünə yadda saxlamaq mümknü deyil. Bunun üçün kifayər qədər yaddaş yoxdur. Lakin üzən nöqtəli növündə olan dəyişən bunu problemsiz özündə saxlaya bilər. C++ dilində bu rəqəmin qısalmış üsulu belə yadda saxlanılır:

1e18

Məqsəd bundadır ki, konkret iş üçün uyğun növdən istifadə etmək lazımdır. 2 və 3 rəqəmlərini siz üzən nöqtəli növündə olan dəyişəndə yadda saxlaya bilərsiniz. Lakin bu çox yer tutur və proqramın işini zəiflədir. Əgər yalnız tam rəqəmli verilənlərdirsə, onda onları tam rəqəmli dəyişəndə yadda saxlamaq olar.

Tam rəqəmli növ double kimi elan olunur, lakin “double” sözünün yerində *int* olmalıdır:

int dəyişənin_adı;

Əgər tam rəqəmli növü dəyişənində üzən nöqtəli verilənlər olarsa, kompilyator onun tam hissəsini yadda saxlayır, qalan hissəsini isə tullayır. Lakin verilənlərin itirilməsi haqqında xəbərbarlıq edir. Misal üçün:

```
int n;  // n adında tam rəqəmli dəyişən elan olunub.
n = 3.7;   //  burada 3 rəqəmi n dəyişəninə yadda saxlanılır, lakin qalan hissə
// (0.7) tullanır.
```

Burada üzən nöqtəli verilənlər növü tam rəqəmli verilənlər növünə çevrilir və **n** dəyişənin yadda saxlanılır. Yenə bir misala baxaq:

```
n = 3.0;   //  Xəbərdarlıq:  double növündən int növünə çevrilmə.
```

Xəbərdarlığa baxmayaraq proqram bu halda düz işləyir, çünki riyaziyyatda 3.0 rəqəmi 3 rəqəminə bərabərdir. Lakin proqramçıların çoxu kompilyatorun xəbərdarlıqlardan xoşları gəlmir. Lakin belə yazsaq:

```
n = static_cast<int>(3.0);
```

Bu halda kompilyator xəbərdarlıq ismarıcını verməyəcək. Lakin bundan yaxşısı tam rəqəmli konstantından istifadə etməkdir. Bu halda xəbərdarlıq olmayacaq, çünki tam rəqəmli növündə olan verilənlər int növündə dəyişəninə yadda saxlanılır:

```
n = 3;
```

Hə, static_cast operatorunun ümumi yazılışı belədir:

static_cast<*növ*>(*ifadə*)

static_cast operatoru parametr kimi *ifadə* qəbul edir və onun çevrilmiş şəklini qaytarır.

Proqramlarda həllərin qəbul edilməsi

Proqramlarda həllırin qəbul edilməsi məhdudlaşdırılıb. Kompüter yalnız dəqiq və konkret instruksiyaları yerinə yetirə bilər.

Bir tərəfdən bu yaxşıdır; lakin digər tərəfdən bu – problem hesab olunur. Xoş xəbər odur ki, kompüter yalnız siz ona dediyiniz işləri dəqiqlikdə yerinə yetirəcək. Problem də odur ki, kompüter **həmişə** siz ona dediyiniz işləri dəqiqlikdə yerinə yetirəcək, bunun nə qədər axmaq olmasına baxmayaraq. Yenə də təkrar edirəm, bu proqramlaşdırmada bir qaydadır – olar bilər ən əsası:

- **Kompüter yalnız konkret başa düşülən instuksiyalar yerinə yetirə bilir.**

Kompüterdə fikirləşmək, anlamaq, gözləmək kimi anlayışlar yoxdur. O ancaq riyazi dəqiq olan əməlləri yerinə yetirə bilir.

if və if-else instruksiyaları

Hərəkəri proqramlaşdırma üçün ən asan üsul deməkdir: "Əgər A doğrudursa, B-ni yerinə yetirmək". if instruksiyası C++ dilində elə bunu yerinə yetirir. Aşağıdakı if instruksiyasının sadə sintaksisi yazılıb:

if *(şərt)*
    instruksiya

Bu instruksiyanın daha çətin sintaksisləri də var, biz yavaş-yavaş çatacayıq. Lakin hələki iki dəyişənlərin bərabərliyini yoxlayan instruksiyaya baxaq:

```
If (x == y)
    cout << "x y-a beraberdir.";
```

Təəccüblüdür. Nümunədə bir bərabər işarəsinin yerinə iki işarə (==) yazılıb. Bu səhv deyil! C++ dilində bu işarələrin iki mənası var: birincisi – bir dəyişənin dəyərinə digərinə kopyalamaq, ikincisi – iki dəyişənin bərabər olub-olmamasını yoxlamaq.

Nə olar ki, bir instruksiya yerinə bir neçəsini yerinə yetirmək? Olar! Lakin bu halda şərtdən sonra { işarəsini və instruksiyaların sonunda } işarələrini yazmaq vacibdir. Bu işarələr *instruksiyalar bloku (statement block)* adlanır:

```
if (x == y)
{
    cout << "x y-a beraberdir" << endl;
    cout << "Bu yaxsidir?" ;
    they_are_equal = true;
}
```

Bu instruksiyanın dəyəri ondan ibarətdir ki, ya bütün instruksiyalar yerinə yetirilir, yaxud heç biri yetirilmir. Necə? Yəni, əgər x dəyişənin dəyəri y dəyişənin dəyərinə bərabər olarsa, onda bütün instruksiyalar yerinə yetirilir, əks halda (bərabər olmasa) heç biri yerinə yetirilmir.

Yenə də if instruksiyasının sintaksisi:

if *(şərt)*
    instruksiya

və

if *(şərt)*
{
    instruksiya (-lar)
}

Həm də əgər şərt yalan olsa, onun instruksiyasını *else* açarlı sözündən istifadə etməklə yazmaq olar:

if *(şərt)*
    instruksiya1
else
    instruksiya2

İndi bizdə iki şərt var, və bu if instruksiyasının tam sintaksisidir. Bir balaca nümunə:

```
if (x == y)
    cout << “x y-a beraberdir”;
else
    cout <<”x y-a beraber deyil”;
```

Bu kodu həm də belə yazmaq olar. Lakin bu vacib deyil:

```
if (x == y)
{
    cout << “x y-a beraberdir”;
}
else
{
    cout <<”x y-a beraber deyil”;
}
```

Əgər {} işarələr arasında olan instruksiyaların sayı 1-dən çoxdursa, onda onları yazmaq vacibdir.

İki operator (= və ==) nəyə lazım?

Əgər siz digər proqramlaşdırma dillərindən (misal üçün Pascal, Basic və s.) istifadə etmisinizsə, onda siz soruşa bilərsiniz: “İki bərabərlik operatorları nəyə lazım, Basic dilində bir operatordan istifadə olunur axı. C++ dilində bərabərlik işarəsi “=” - dır, Pascal dilində “:=” – dır və s. if instruksiyasına (şərtində) nəyisə nəyəsə bərabər etmək olmaz, yalnız birinin digərinə bərabər olub-olmamasını yoxlamaq olar. Misal üçün:

```
if (x = y)  // SƏHVDİR!  Bərabər olunur!
        cout << “x y-a beraberdir”;
```

Misal 2.1. Cüt ya tək?

Yaxşı, girişə son qoyaq. Vaxt gəlib əsl hazır proqrama baxmaq, hansı ki, həllərin qəbulundan istifadə edir. Bu nümunə bizə yeni operator (%) ilə tanış edəcək və if-else instruksiyalarının işdə görsədəcək.

Kod 1.6----------

```
#include <iostream>
using namespace std;
int main()
{
        int n, remainder;
// Klaviaturada yığılan rəqəmi götürmək.
        cout << “Reqem yigin ve ENTER duymesine basin: ”;
        cin >> n;
// 2-ye bölünməsindən qalığı tapmaq.
        remainder = n %2;
// Əgər remainder dəyişənin dəyəri 0-a bərabərsdirsə,
// yığılan rəqəm cütdür.
        if (remainder == 0)
                cout << “Reqem cutdur”;
        else
                cout << “Reqem tekdir”;
```

```
return 0;
}
```

Yenə də təkrar edirəm - // işarələrindən sonra gələn mətn (işarə ilə birlikdə) vacib deyil. Mən sadəcə sizin üçün kodun rahat oxunması üçün yazmışam.

Bu necə işləyir

Birinci instruksiya **n** və **remainder** adlarında iki dəyişən elan edir.

```
int n, remainder;
```

Sonra, istifadəçi klaviaturada yığıdığı rəqəm **n** dəyişəninə yadda saxlanılır. Əsl vaxtda bu belə şəkildə olmalıdır:

```
cout << “Reqem yigin ve ENTER duymesine basin:  “;
    cin >> n;
```

İndi isə yoxlamaq lazımdır ki, **n** dəyişənində olan rəqəm cütdür yoxsa tək. Bunu necə etmək? Cavab: bu rəqəmi 2-yə bölmək və qalığına baxmaq. Əgər qalıq 0-a bərabərdirsə, onda rəqəm cütdür, əks halda rəqəm təkdir. Bu misaldakı kimidir. Növbəti instruksiya **n** –ni 2-yə bölür və qalığını alır. Bu əməliyyat *modul ilə bölünmə (moduls)* və yaxud *bölünmədən qalıq (remainder)* adlanır. Bölünmənin nəticəsi **remainder** dəyişəninə saxlanılır.

```
remainder = n % 2;
```

C++ dilində faiz işarəsi (%) öz adi dəyərini itirir və bunun yerində bölünmədən alınan qalığı tapır.

n dəyişənini 2-yə böldükdə, qalıqda (yəni cavabda) biz 0 (cüt rəqəm) və yaxud 1 (tək rəqəm) alırıq. if instruksiyası isə qalığı 0 ilə bərabərliyini yoxlayır və uyğun ismarıcı göstərir.

```
if (remainder == 0)
    cout << “Reqem cutdur”;
else
    cout << “Reqem tekdir”;
```

Ona diqqət yetirin ki, burada iki bərabərlik işarəsi (==) istifadə olunmuşdur. Bu iki işarə bir işarə kimi saylır (bu halda). Mən bunu sizə dəfələrlə deyirəm, çünki C++ ilk dəfə öyrənəndə özüm çoxlu sayda belə səhvlər etmişdim!

Həm də bu instruksiyanı belə yazmaq olar:

```
if (remainder == 0)
{
        cout << “Reqem cutdur”;
}
else
{
        cout << “Reqem tekdir”;
}
```

Kodun optimallaşdırılması

İndi yuxarıda göstərdiyimiz proqramın belə versiyası effektiv deyil. Orada heç **remainder** dəyişəni də vacib deyil. Aşağıdakı versiya ondan yaxşıdır:

Kod 1.7----------

```
#include <iostream>
using namespace std;
int main()
{
        int n;
// Klaviaturada yığılan rəqəmi götürmək.
        cout << “Reqem yigin ve ENTER duymesine basin: ”;
        cin >> n;
// 2-ye bölünməsindən qalığı tapmaq.
// Əgər remainder dəyişənin dəyəri 0-a bərabərsdirsə,
// yığılan rəqəm cütdür.
        if (n % 2 == 0)
                cout << “Reqem cutdur”;
        else
                cout << “Reqem tekdir”;
return 0;
}
```

Bu versiyada modul ilə bölünmə elə şərtin daxilində gedir, alınan cavab 0 ilə bərabərliyi yoxlanılır.

Məsələlər

Məsələ 2.1.1. *Proqram yazın ki, o yığılan rəqəmin 7-yə bölünüb-bölünməməsini yoxlasın və uyğun ismarıc versin. (Kömək: əgər rəqəm 7-yə qalıqsız bölünürsə, bu o deməkdir ki, rəqəmi 7-yə bölmək olar və qalıqda 0 alınır.)*

Dövrələrə giriş

İstənilən proqramlaşdırma dillərində ən güclü konsepsiya dövrə sayılır. Bu bölmədə siz görəsəksiniz ki, bir neçə kod sətri ilə instruksiyaların minlərlə dəfə etmək olar.

Proqram dövrədə olan zaman, dövrənin şərti düz olana kimi yerinə yetirilir. Klassik forma *while* göstərmək olar:

while *(şərt)*
    instruksiya

if instruksiyasında olduğu kimi, burada da siz instruksiyanın (-ların) əvvəl və sonra {} işarələrini yaza bilərsiniz:

while *(şərt)*
{
    instruksiya
}

if instruksiyasında olduğu kimi, while instruksiyasında da, əgər şərt doğrudursa instruksiya yerinə yetirilir, əks halda yetirilmir. Lakin while instruksiyasında instruksiyalar sonsuz sayda yerinə yetirilir.

Daha aydın desək, proqram bütün instruksiyaları yerinə yetirdikdən sonra yenidən şərti yoxlayır. Belə, while instruksiyası if və goto instruksiyalarının köməyi ilə görsədilə bilər.

İş belə gedir:

- Şərti yoxlamaq. Əgər şərt doğrudursa 2, 3 addım yerinə yetirmək. (Əks halda, biz qurtardıq; dövrənin axırı üçün birinci instruksiyaya qayıtmaq.)
- İnstruksiyanı yerinə yetirmək.
- 1-ci addıma qayırmaq.

Ən asan nümunə - konsola 1-dən N-ə kimi rəqəmləri çap etmək, harada ki N - dəyişəndir. Bu misala psevdokod ilə baxaq, bu o deməkdir ki nümunə hələ ki azərbaycan dilində olacaq.

C/C++ dillərinin proqramçıları razılaşma ilə dəyişənlərin adlarını balaca hərflə - "**N**" əvəzində "**n**" yazırlar. Lakin bu, C/C++ dillərini qaydası deyil, dəyişənlərin adlarını həm böyük, həm də kiçik hərflə yazmaq olar.

Proqramın əvvəlində dəyişənləri elan etmək lazımdır. Tutaq ki, **N** və **İ** tam rəqəmli dəyişənlərdir.

Rəqəmləri bax belə çap etmək olar:

1. Klaviaturadan yığdığı rəqəmi **N** dəyişəninə yadda saxlamaq.
2. **İ** dəyişənin dəyərini 1 edək.
3. **İ** dəyişənin dəyəri **N** dəyişənin dəyərinə bərabər və ya ondan kiçik olana kimi etmək
    a. **İ** dəyişənin dəyərini konsola çap etmək.
    b. **İ** dəyişənin dəyərinin üzərinə 1 rəqəmini əlavə etmək.

Sonsuz dövrələr

Dövrənin elə şərtini elə yazmaq olar ki, o həmişə doğru olsun? Bu mümkün olarsa, nə baş verər? Cavab – (1) bəli, bu proqramlaşdırmanın populyar səhvidir; və (2) dövrə, kompüter öz gücünü itirənə kimi yerinə yetiriləcək.

Misal 2.2. 1-dən N-ə kimi rəqəmlərin çapı

İndi isə, yuxarıda danışdığımız proqramı yaratmaq üçün C++ dilinin instruksiyalarından istifadə edək. Bunun üçün bir *while* instruksiyası və ondan sonra iki instruksiya yazmaq lazım gələcək.

Kod 1.7----------

```
#include <iostream>
using namespace std;
int main()
{
	int I, N;
	cout << “Reqem yigin ve ENTER duymesine basin: “;
	cin >> N;
	I = 1;
	while  (I <= N)
{
	cout << I << “ “ ;
	I = I + 1;
}
	return 0;
}
```

Proqram başlananda yığdığınız rəqəmə kimi sayacaq. Misal üçün, əgər siz 6 rəqəmini yığmısınızsa, onda:

```
1  2  3  4  5  6
```

olacaq.

Bu necə işləyir

Bu nümunədə yeni operator yazılmışdır, lakin mən əminəm ki siz onun nə etdiyini artıq anlamısınız. Bu “kiçikdir-bərabərdir” operatorudur:

İ <= n

“kiçikdir-bərabərdir” operatoru (<=) yoxlayan operatorlardan biridir, hansı ki, iki dəyər qaytarır – **true** (doğru), **false** (sehv).

Operator	Nə edir
==	Bərabərliyi yoxlayır
!=	Bərabərsizliyi yoxlayır
>	Böyüklüyünü yoxlayır

<	Kiçikliyini yoxlayır
>=	Böyük və bərabərliyi yoxlayır
<=	Kiçik və bərabərliyi yoxlayır

Əgər siz “Dövrələrə giriş” bölməsində məntiqi izləmisinizsə, onda özünə görə dövrələr çətin deyil. Fiqurlu mötərizələr ({ }) while instruksiyasının sonra birdən çox instruksiya yerinə yetirməyə imkan yaradır.

```
while  (I <= N)       // İ dəyişəninin dəyəri N-dən az və bərabər olduğu zaman
{
        cout << I << " " ;   // Çap etmək
        I = I + 1;           // İ dəyişəninin dəyərini
}
```

Dövrədə ilk instruksiya:

```
cout << I << " " ;   // Çap etmək
                     // İ dəyişəninin dəyərini
```

Bu instruksiya İ dəyişəninin dəyərini çap edəndən sonra probel yerləşdirir. Ona görə də rəqəmlər arasında ara məsafə var:

```
1   2  3  4  5  6
```

belə yox:

```
12345
```

Bundan sonra, növbəti dövrə instruksiyası yerinə yetirilməzdən əvvəl İ dəyişəninə 1 rəqəmi əlavə olunur. Bu zəmanət verir ki, dövrə nə vaxtsa öz işini bitirəcək, çünki İ dəyişənin dəyəri N dəyişənin dəyərindən böyük olacaq.

```
İ = İ + 1;
```

Məsələlər

Məsələ 2.2.1. *Elə proqram yazın ki,* ***n1*** *və* ***n2*** *dəyişənlərində bütün rəqəmləri çap etsin, harada ki,* ***n1*** *və* ***n2*** *– istifadəçi tərəfdən yığdığı iki rəqəmdir.(Kömək: yığmaq üçün iki sətir olmalıdır, sonra* ***İ*** *dəyişənin dəyərini* ***n1*** *dəyişənin dəyərinə bərabər edin, şərtdə isə* ***n2*** *dəyişənini istifadə edin.)*

Məsələ 2.2.2. *Nümunəni elə dəyişdirin ki,* ***n****-dən 1-ə kimi rəqəmlər tərsinə(5 4 3 2 1) çap olunsun. (Kömək: şərtin daxilində I = I – 1; istifadə edin.)*

C++ dilində true və false dəyərləri

"**true**" və "**false**" dəyərləri əslində nədir? Digər dəyərlər kimi, bunlar da kompüterdə rəqəmsal kimidir?

Əlbəttə ki, bu elədir. Hər bul operatoru (yəni məntiqi yoxlama operatoru) 0 və yaxud 1 qaytarır.

Şərtin yoxlanma nəticəsi	İfadə qaytarır
true	1
false	0

Həm də istənilən sıfırlı olmayan dəyər ona yönəlir ki, şərt doğru sayılır (true). Ona görə də aşağıdakı nümunədə instruksiya həmişə yerinə yetirilir:

```
// Həmişə yerinə yetirilir!!!
if (1)
{
        // İnstruksiya yerinə yetirmək
}
```

Növbəti nümunədə əvvəl dediyim kədərli dövrə alınır, hansı ki heç vaxt öz işini bitirmir:

```
// Həmişəlik dövrə
while (1)
{
        // Instruksiya yerinə yetirmək
}
```

Bütün yoxlama operatorları 0 və 1 qaytardığına baxaraq, siz tam rəqəmli dəyişəni elan edib bullu "bayraq" kimi istifadə edə bilərsiniz – dəyişən, hansı ki, "true" və "false" dəyərlərini saxlayır. Misal üçün:

```
int is_less_than;
is_less_than = (i < n);  // "true" (1) dəyərini saxlamaq, əgər İ dəyişənin
//  dəyəri N dəyişənin dəyərindən azdır.
```

is_less_than dəyişəni özündə şərtin yoxlama nəticəsini saxlayır.

bool verilənlər növü

C++ dilinin ən axırıncı versiyaları xüsusi bool növü dəstəkləyir (“Boolean” – məntiqi), hansı ki özündə iki tam rəqəmli dəyər saxlaya bilər – 0 və 1. Yəni **true (1)** – doğru, **false (0)** – səhv. Əgər sizin kompilyatorunuz bool növünü dəstəkləyirsə, onda onu istifadə etmək int növünü istifadə etməkdən qat-qat məsləhətlidir:

```
bool is_less_than;       // “true” (1) dəyərini saxlamaq, əgər i dəyişənin
is_less_than = (i < n);  //  dəyəri n dəyişənin dəyərindən azdır.
```

++ operatoru

C dilinin istehsalçıları qısaltma etmək istəyirdilər. Proqramçılar arasında ən sevimli qısaltma ++ operatorudur. Bu operator dəyişənin üzərinə 1 rəqəmini əlavə edir. Misal üçün:

```
n ++;      // n = n + 1
```

Əvvəl görsədilmiş dövrəyə baxaq:

```
        while  (I <= N)
{
        cout << I << “ “ ;
        I = I + 1;
}
```

Dövrədə ikinci instruksiya aşağıdakı ilə əvəz oluna bilər:

```
I++;
```

Tam olaraq:

```
        while  (I <= N)
{
```

```
        cout << I << " " ;
        I++;
}
```

Bu əvəzetmək bizi artıq əziyyətdən qurtardı. Lakin hər şey yaxşılaşacaq. Dövrə aşağıdakı kimi qısaldıla bilər:

```
                while  (I <= N)
{
        cout << I++ << " " ;
}
```

Biz görürük ki, dövrənin daxilində cəmi bir instruksiya var, onda görə də onu yenə də aşağıdakı kimi qısaltmaq olar:

```
while  (I <= N)
        cout << I++ << " " ;
```

Siz maraqlana bilərsiniz, çıxmaq (-1) üçün buna oxşar operator var? Əlbəttə var.

Operator	Hərəkət
var++	*var* dəyişəninin hazırki dəyərini qaytarır, sonra üzərinə 1 rəqəmi əlavə edir.
++var	*var dəyişənin üzərinə 1 rəqəmini əlavə edir, sonra onun dəyərini qaytarır.*
var--	*var* dəyişənin hazırki həyərini qaytarır, sonra ondan 1 rəqəmini çıxır.
--var	*var* dəyişənin dəyərindən 1 rəqəmini çıxır, sonra onun dəyərini qaytarır.

İfadələr ilə yoxlama instruksiyaları

Bu vaxta kimi mən "instruksiya" (statement) və "ifadə" (expression) kimi terminlər işlətmişəm. Bu terminlər C++ dilinin fundamentalllarıdır, onda görə də onların nə olduğunu bilmək vacibdir.

Burada müəyyənlik qoymaq çətindir. Lakin bunun haqqında inamla demək olar: (1) C++ dilində proqram bir və ya bir neçə instruksiyadan ibarətdir, və bu funksiya (2) bir və ya bir neçə funksiyadan ibarətdir. Ümumi şəkildə desək siz instruksiyanı ";" işarəsi ilə müəyyən edə bilərsiniz. Bu da misal:

```
cout << i++ << " ";
```

C++ dilində sadə instruksiya bir sətirdən ibarətdir. Yəni, “;” işarəsi instruksiyanın bitirilməsini deyir. Ona görə də bir neçə instruksiyanı bir sətirdə yerləşdirmək olar (lakin məsləhət görülmür):

```
cout << I << “ “; I++;
```

Yaxşıdır ki – siz deyərsiz. Bəs ifadə nədir? Bu sıx bağlanmış konsepsiyalardır, lakin çox vacibdirlər. Aşağıda nümunə kimi bir neçə ifadə yazılıb:

```
x          // x dəyişənin dəyərini qaytarır
12         // 12 qaytarır
x + 12     // x + 12 ifadəsinin nəticəsini qaytarır
x == 13  //  bərabərliyi yoxlayır: 0 və ya 1 qaytarır
x = 13     // Bərabər etmə: x dəyişənin dəyərini 13 edir.
num++  // qısaltmadan qabaq num dəyişənin dəyərinin qaytarır
i = num++ +2  // Çətin ifadə: i dəyişənin yeni dəyərini qaytarır
```

Bulev məntiqinə giriş

Hərdən çətin ifadələri yazmaq üçün “və”, “və yaxud” sözlərindən istifadə olunur. Misal üçün aşağıda (ana dilində) şərt “və” ilə istifadə olunub:

```
Əgər age > 12 və age < 20
                Adam yeniyetmə sayılır.
```

“və” sözü ilə şərti ifadə etmək üçün İBM proqramçıları *Bulev cəbrindən (Boolean algebra)* istifadə edirlər, hansı ki, IXX əsrin riyaziyyatçısı olan Corc Bulun adına qoyulub. Bulev cəbri, tələb olunan işi görür.

Aşağıdakı cədvəldə C++ dilinin üç bulev (məntiqi) operatorları yazılıb:

İşarələr	**Əməliyya**	**C++ dilinin sintaksisi**	**Hərəkət**
&&	AND	expr1 && expr2	**expr1** və **expr2** ifadələri həll edir. Əgər ifadələrin ikisinin nəticəsi **true**-dirsə, onda **true** qaytarır, əks halda **false.**
||	OR	expr1 || expr2	**expr1** və **expr2** ifadələri həll edir.

			Əgər ifadələrdən heç olmasa birinin nəticəsi **true**-dirsə, onda **true** dəyəri qaytarılacaq, əks halda **false.**
!	NOT	! expr1	**expr1** ifadəni həll edir. Əgər nəticə 0-a bərabərdirsə, onda **true** qaytarır, əks halda **false.**

Buna baxaraq, əvvəlki nümunəni belə yazmaq olar:

```
if (age > 12 && age < 20)
        cout << “Bu adam yenitemdedir”;
```

Misal 2.3. İnsan yaşının yoxlanılması

Bu nümunədə “və” (&&) operatorunun sadə şəkildə istifadə olunması görsədilib.

Kod 1.8----------

```
#include <iostream>
using namespace std;
int main()
{
        int n;
        cout << “Yasinizi daxil edin ve ENTER duymesine basin: “;
        cin n;
        if (n > 12 && n < 20)
                cout << “Siz yeniyetmesiniz”;
else
                cout << “Siz yeniyetme deyilsiniz”;
        return 0;
}
```

Bu necə işləyir

Bu böyük olmayan proqram iki əməliyyatdan ibarət şərt istifadə edir:

n > 12 && n < 20

"&&" işarənin prioriteti, ">", "<" işarələrinə nisbətən azdır, onda görə də əvvəlcədən kənarda olan əməliyyatlar yerinə yetirilir. Bu, aşağıdakına ekvivalentdir:

(n > 12) && (n < 20)

Məsələlər

Məsələ 2.3.1. *Proqram yazın ki, 0 ilə 100 arasında rəqəmləri doğru, digər rəqəmləri isə səhv hesab etsin. (Kömək:* **n > 0 && n < 100**).

Riyazi kitabxanasına giriş

İndiki vaxta kimi mən, C++ dilinin standard kitabxanasında istifadə etmişəm. Ona görə də aşağıdakı sətri yazmaq lazım idi:

#include <iostream>

İndi isə mən, riyazi funksiyalarından istifadə edəcəm. C++ dilinin istənilən riyazi operatorunu (+,*, - və %), çünki onlar standard kitabxanaya aiddirlər. Lakin hər hansı xüsusi riyazi funksiyalarından istifadə etmək üçün aşağıdakı sətri yazmaq lazımdır:

#include <math.h>

Riyazi funksiyalar özündə triqonometrik funksiyalar (**sin, cos, tan, asin, acos, atan** və s.), loqarifmik funksiyalar (**log, log10**), eksponesial funksiyalar (**exp, pow**) və digər yaxşı şeylər daşıyır. Bu bölmədə yalnız bir riyazi funksiya istifadə olunub: **sqrt** kvadrat kökü.

```
#include <math.h>
//...
double x;
x = sqrt(2.0);
```

Əgər nəticəni tam rəqəmli verilənlər növünə yadda saxlasaq, onda ədədin tam hissəsi yadda saxlanılacaq, qalan hissəsi isə tullanılacaq:

```
int x;
n = sqrt(2.0);
```

Nəticədə n dəyişəninə 1 rəqəmi yadda saxlanılır. Əsl cavab 1.41421 – dir. Bu ədəd 1 rəqəminə kimi yuvarlaqlaşdırılır və n dəyişəninə yadda saxlanılır.

Siz həm də təəccüblənə bilərsiniz: **iostream** və **math.h.** Fərq bundadır ki, iostream virtual fayldır, hansı ki, kompilyatorun içərisinə daxildir. Lakin math.h isə real fayldır, hansı ki, riyazi funskiyalarından istifadə etmək üçün əlavə olunur.

Misal 2.4. Sadə rəqəmə yoxlanılması

Indi bizim əlimizdə C++ dilinin kifayət qədər alətləri var. Ona görə də biz nəsə maraqlı və faydalı bir şey yarada bilərik. İndi biz müəyyən edəcəyik – hansı rəqəm sadədir, hansı isə mürəkkəb. Bu da kodun özü:

Kod 1.9----------

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
        int n;     // yoxlanılan rəqəm
        int i;     // dövrələrin sayı
        int is_prime;   //  bul bayraq

 //  Əksini sübut etməyənə kimi rəqəmin sadə olduğunu tutaq.

is_prime = true;

//  Klaviaturadan rəqəmi tutmaq

cout << “Reqem yigin ve ENTER duymesine basin: “;
cin >> n;

i = 2;
while (i <= sqrt(static_cast<double>(n)))
{
if (n % i == 0)
```

```
is_prime = false;
i++;
}
if (is_prime)
        cout << “Reqem sadedir”;
else
        cout << “Reqem murekkebdir”;

        return 0;
}
```

Bu necə işləyir

Proqramın nüvəsi dövrə sayılır. Diqqət yetirin: verilənlərin çevrilməsi prosesində problemlər olmasın deyə, tam rəqəmli **n** növü double növünə çevrilməlidir, bu vacibdir, çünki funksiya yalnız double formatlı verilənləri alıb-qaytarır.

```
        while (i <= sqrt(static_cast<double>(n)))
        {
        if (n % i == 0)
        is_prime = false;
        i++;
        }
```

Gəlin buna daha aydın baxaq. Aşağıda bunun ana dilində yazılmış versiyası görsədilib:

i dəyişənin dəyərini 2 etmək.
i dəyişənin dəyəri **n** dəyişənin dəyərinin kvadrat kökündən böyük və ya kiçik olan zaman
        əgər **i** dəyişənin dəyəri **n** dəyişənin dəyərinə qalıqsız bölünürsə
                n dəyişənin dəyəri mürəkkəb ədəddir.
i dəyişənin dəyərinə 1 əlavə etmək.

Xülasə

Bu başlıqda aşağıdakıları öyrənmişik:

- ✓ Hər hansı iş üçün uyğun verilənlər növünün istifadə olunması. Heç vaxt kəsri olmayan ədədlər int verilənlər növündə yerləşməlidir.
- ✓ Tam rəqəmli verilənlər növündə olan dəyişəni elan etmək üçün int, ondan sonra dəyişənin adını yazmaq lazımdır:

  int *deyisenin_adi***;**
- ✓ C++ dilində həllərin qəbul edilməsinin ən sadə üsulu if instruksiyasının istifadə olunmasıdır:

  if *(şərt)*

  instruksiya
- ✓ if instruksiyasında əks hal üçün else sözündən də istifadə edə bilərsiniz:

  if *(şərt)*

  instruksiya

  else

  instruksiya
- ✓ İstənilən istruksiya istifadə olunan yerdə "mürəkkəb" instruksiyadan istifadə edə bilərsiniz ({} işarələrini artırmaqla):

  if *(şərt)*

  {

  instruksiya

  }
- ✓ Bərabər edən operatorunu (=) bərabərliyi yoxlayan (==) operator ilə səhv salmayın. == operatoru iki dəyəri yoxlayır, yəni əgər bərabərsidrsə **true (1)**, bərabər deyilsə **false (0)** qaytarır. Misal üçün:

  if (x == y)

  is_equal = true;
- ✓ while instruksiyası, instruksiyanı və yaxud mürəkkəb instruksiyanı çoxlu sayda yerinə yetirir. Lakin bu instruksiyanın da öz şərti var, yəni əgər şərt doğrudursa dövrə yerinə yetirilir, əks halda yetirilmir.

  while *(şərt)*

  instruksiya
- ✓ Modul ilə bölünmə operatoru iki dəyəri bölür və qalığı qaytarır:

  13 % 5
- ✓ İnstruksiya (çox hallarda) – C++ dilinin ";" işarəsi ilə bitən sətirdir. Bundan başqa bir neçə instruksiyalar kiçik instruksiyalardan ibarət ola bilər.
- ✓ İfadə instruksiyaya çevrilə bilər. Misal üçün:

  num++
- ✓ Dəyərin üzərinə 1 rəqəmini gəlmək üçün qısa operatordan istifadə edə bilərsiniz. Belə giriş proqramın kodunu daha optimallaşdırır:

  cout << n++;
- ✓ Çətin şərtlər yaratmaq üçün bulev operatorlardan istifadə etmək olar:

Və (&&), və yaxud (||) və deyil (!).

BAŞLIQ 3

Rahat bə universal “for” instruksiyası

Bir neçə məsələlər elə ümumidir ki, C++ dili, onları həll etmək üçün xüsusi sintaksis bağışlayır. Buna misal olaraq ++ operatorunu göstərmək olar.

Digər bir misal for instruksiyasıdır. Siz görəcəksiniz ki, bir neçə nümunədən sonra bu instruksiyadan siz həmişə istifadə edəcəksiniz. Mən bunun üçün bütün başlıq ayırmışam.

Say üçün istifadə olunan dövrə

Dövrələr ilə işləməklə biz fikir verdik ki, while instruksiyasının məqsədi – müəyyən olunmuş saya qədər hər hansı instruksiyaların yerinə yetirilməsidir. Misal üçün:

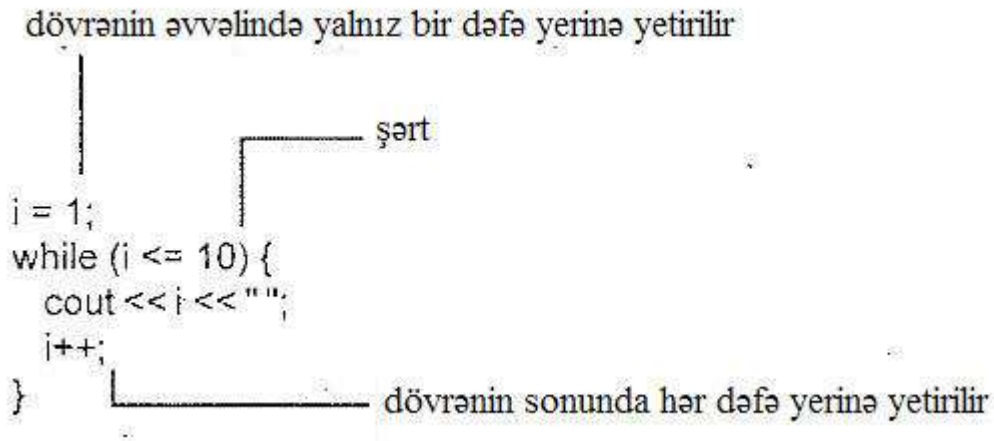
```
i = 1;
while (i <= 10)
{
        cout << i << “ “;
        i++;
}
```

Bütün deyilənlərdən və edilənlərdən sonra, kompüter sayı 1-dən 10-a kimi yetirəndə nə baş verir? Bu elə, kompüterlərin ən yaxşı bacardığı şeydir. Dövrənin dəyişəninin dəyəri 1 olur, sonra hər dövrənin sonunda 1 rəqəmi ilə artır. Bunu söz ilə başa salmaq olar:

- i dəyişənin dəyərini 1 etmək.
- Dövrəni yerinə yetirmək.
- i dəyişənin dəyərini 2 etmək.
- Dövrəni yerinə yetirmək.

- i dəyişənin dəyərini 3 etmək.
- Dövrəni yerinə yetirmək.
- i dəyişənin dəyəri 10 olana kimi bunların hamısını təkrar etmək.

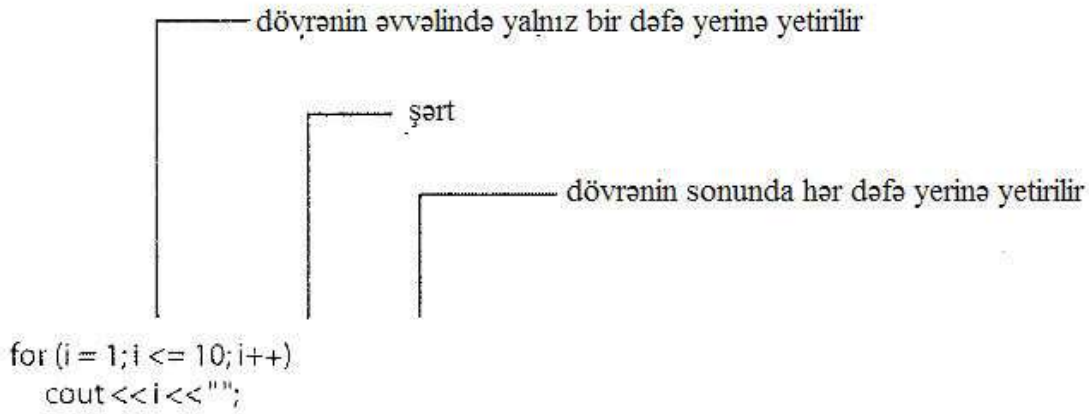
Digər sözlə, hər dəfə i dəyişənin dəyərinə 1 əlavə etməklə dövrəni yerinə yetirmək. Dövrələrin belə şəklində müəyyən hərəkətlər həmişə yerinə yetirilir. Biz belə hərəkətləri qeyd edə bilərik:



Yaxşı olardı ki, bütün bunları bir sətirdə yazmaq.

“for” dövrəsinə giriş

for instruksiyası elə bu mexanizmini bağışlayır:



Bu nəinki sıx, həm də səliqəlidir.Dövrənin bütün parametrləri mötərizələrdə yazılır. Yenədə for instruksiyasına diqqət yetirin:

```
for (i = 1; i <= 10; i++)
        cout << i << " ";
```

Əqli cəhətdən bu instruksiyanı deşifrələməklə, mötərizədə olan üç əsas şeyi yaddan çıxartmayın – *initializtor*, *şərt* və *inkrement*.

- **i = 1** ifadəsi initialozatordur. O, dövrənin əvvəlində cəmi bir dəfə yerinə yetirilir. Bu halda i dəyişəni birinci dəyəri olan 1 alır.
- **i <= 10** ifadəsi şərt sayılır.
- **i++** ifadəsi inkrement sayılır. O, dövrənin sonunda yerinə yetirilir.

for instruksiyasına ekvivalent olan while instruksiyasına qayıdaq:

```
i = 1
while (i <= 10)
{
cout << i << " ";
i++;
}
```

Mən müəyyən etmişəm ki, bir çox nümunələrə diqqət yetirməyənə kimi sizə for instruksiyası hələ qaranlıq qala bilər (aydın olmamaya bilər).

Çoxlu nümunələr

Mən, siz gördüyünüz nümunənin dəyişməsindən başlayacam. Dövrənin i dəyişəni 1 dəyəri ilə intializasiya olunur və dövrə, şərtin nəticəsi **false (0)** olandan sonra öz işini bitirəcək.

```
for (i = 1; i <= 5; i++)
        cout << i << " ";
```

Cavabında belə bir şey alırıq:

```
1   2  3  4  5
```

Növbəti dövrə 10-dan 20-yə kimi yerinə yetirilir:

```
for (i = 10; i <= 20; i++)
        cout << i << " ";
```

Nəticədə alacayıq:

10 11 12 13 14 15 16 17 18 19 20

Bu halda initialoztor **i = 10** ifadəsi olur, şərt isə **i <= 20.**

Parametrlərin konstant olması vacib deyil. Növbəti nümunədə onlar dəyişənlə müəyyən olunur.

```
n1 = 32;
n2 = 38;
for (i = n1; i <= n2; i++)
        cout << i << " ";
```

Nəticədə alacayıq:

32 33 34 35 36 37 38

Bu proqramın tam kodu belə olur:

Kod 2.0----------

```
#include <iostream>
using namespace std;
int main()
{
int i, n1, n2;
n1 = 32;
n2 = 38;
for (i = n1; i <= n2; i++)
        cout << i << " ";
return 0;
}
```

Nəticəsi isə MS-DOS əməliyyat sistemində:

İnkrement ifadəsi istənilən ola bilər, vacib deyil ki, yalnız i++ olsun. Misal üçün, dövrənin əks tərəfə rəqəmləri saysın deyə i—ifadəsindən istifadə edə bilərik. Ona da diqqət yetirin ki, bu halda şərt “>=” olur.

```
for (i = 10; i >=1;  i--)
        cout << i << “ “;
```

Nəticədə alırıq:

```
10  9  8  7  6  5  4  3  2  1
```

for instruksiyası çox çevikdir. İnkrementin ifadəsini dəyişməklə sayın addımını 2-yədək çoxaltmaq olar:

```
for (i = 1; i <= 11; i = i +2)
        cout << i << “ “;
```

Nəticədə alacayıq:

```
1  3  5  7  9  11
```

Aşağıdakı nümunə isə göstərir ki, dövrənin dəyişəni kimi yalnız **i –** ni istifadə etmək gərək deyil, onun yerində misal üçün **j** adında dəyişəni də istifadə etmək olar:

```
for (j = 1; j <= 5; j = j ++)
        cout << j * 2 << “ “;
```

Nəticədə alacayıq:

2 4 6 8 10

Diqqət yetirin ki, bu nümunədə ekrana **j * 2** nəticəsi çıxır. Ona görə də cavab həmişə cüt rəqəm olacaqdır.

Misal 3.1. for instruksiyasında istifadə etməklə 1-dən N-ə kimi rəqəmlərin çapı

İndi isə biz, for instruksiyasını hazır proqramda istifadə edək. Bu, misal 2.2 kimi iş görür: 1-dən **n** dəyişənin dəyərinin diapazonuna kimi bütün rəqəmləri çap edir. Lakin bu versiya daha kompaktdır.

Kod 2.0----------

```
#include <iostream>
using namespace std;
int main()
{
int i, n;
        cout << “Reqem yigin ve ENTER duymesine basin: “;
cin >> n;
for (i = 1; i <= n; i++)
        cout << i << “ “;
return 0;
}
```

Misal üçün, əgər istifadəçi 9 rəqəmini yığıbsa, nəticə belə olacaq:

1 2 3 4 5 6 7 8 9 10

Bu necə işləyir?

Bu nümunədə sadə dövrə istifadə olunub. Əvvəlkindən fərqi odur ki, burada maksimal rəqəmi istifadəçi özü yazır.

```
        cout << “Reqem yigin ve ENTER duymesine basin: “;
        cin >> n;
```

Dövrə 1-dən n-ə kimi bütün rəqəmləri çap edir, harada kı, n – yığılan rəqəmdir.

```
        for (i = 1; i <= n; i++)
```

cout << i << " ";

Təkrar üçün:

- ✓ **i = 1** ifadəsi initializaotor ifadəsidir; o dövrədən qabaq yalnız bir dəfə yerinə yetirilir. Belə şəkildə, **i** dəyişənin başlanğıc dəyəri 1 olur.
- ✓ **i <= n** ifadəsi şərt hesab olunur. O hər dövrənin generasiyasında yoxlanılır – şərt doğrudursa (**true (1)**) instruksiyalar yerinə yetirilsin, əks halda (**false (0)**) yetirilməsin.
- ✓ **i++** ifadəsi inkrement adlanır. Bu ifadə hər dövrənin sonunda yerinə yetirilir.

Beləliklə, proqramın məntiqi belədir:

i dəyişənin dəyərini 1 etmək
i dəyişənin dəyəri **n** dəyişənin dəyərinə bərabər və yaxud ondan kiçik olan zaman
i dəyişənin dəyərini çap etmək.
i dəyişənin dəyəri üzərinə 1 əlavə etmək

Məsələlər

Məsələ 3.3.1. *for instruksiyasından istifadə etməklə elə proqram yazın ki,* ***n1****-dən* ***n2****-yə kimi bütün rəqəmləri çap etsin, harada ki n1 və n2 istifadəçinin yığdığı rəqəmlərdir. (Kömək: n1 və n2 yığmaq üçün sətirlər olmalıdır. Sonra for instruksiyasının daxilində i dəyişənini initializasiya etmək (i = n1)və n2 dəyişənini dövrənin şərtində istifadə etmək laızmdır.,)*

Məsələ 3.3.2. *Nümunəni elə dəyişdirin ki, 1-dən 5-ə kimi rəqəmlər əks tərəfə sayılsın (5 4 3 2 1).*

FOR instruksiyası istifadəsi zamanı bloklar instruksiyası

Bu vaxta qədər mən hər dövrün daxilində aşağıdakı instruksiyadan istifadə edirdim:

cout << i << " ";

Əlbəttə ki, siz bu instruksiyadan istifadə etməyə məcbur deyilsiz. Siz i dəyişənin dəyərini çap etməməlisiniz; əslində, siz nə isə çap etməyə məcbur deyilsiniz. Mən bu variantı sizə dövrün necə işləməsini göstərmək üçün seçmişəm. Siz dövrlərdə digər müxtəlif şeylər edə bilərsiniz.

if və **while** instruksiyalarında olduğu kimi, **for** instruksiyasında da blok instruksiyalardan istifadə etmək olar:

for (*başladan; şərt; artırıcı***) {**
    instruksiya
}

Aşağıda 2 instruksiyadan ibarət for instruksiyası verilmişdir:

```
for (i = 1; i <= 10; i++) {
        cout << “The square root of  “ << i << “ is ”;
        cout << sqrt(i) << endl;
}
```

Aşağıdakı kod yuxarıdakı koda ekvivalentdir:

```
i = 1;
while (i <= 10) {
        cout << “The square root of  “ << i << “ is ”;
        cout << sqrt(i) << endl;
        i++;
}
```

Dövrün dəyişənlərinin dinamik elan edilməsi

for instruksiyasının yaxşı cəhətlərdən biri də odur ki, siz onun dəyişəninin elə onun özündə elan edə bilərsiniz, harada ki, həmin dəyişən yalnız dövr üçün görünən olacaq. Dəyişən “dinamik” olaraq for dövrünün daxilində tez istifadə etmək üçün yaradılır. Misal üçün:

```
for (int i = 1; i <= 10; i++)          // i = 1 n-ə qədər üçün,
        cout << i << “ “;              // i dəyişənin dəyərinin çap olunması
```

Bu nümunədə i dəyişəni for dövrün daxilində elan edilib və istifadə olunur.

Kod 2.1----------

```
#include <iostream>
using namespace std;

int main() {
	int n;

// Klaviaturadan rəqəm oxutmaq

	cout << “Enter a number and press  ENTER:  “;
	cin >> n;

	for (int i = 1; i <= n; i++) // i=1 n-ə qədər üçün
		cout << i << “ “;    //  i dəyişənin dəyərinin çap edilməsi

	return 0;
}
```

Misal 3.2. for dövründən istifadə etməklə sadə rəqəmə yoxlanılması

Bu bölmədə mən 2.4 – cü misala qayıdıb o misalın *while* instruksiyanı yerinə *for* instruksiyasından istifadə etməklə necə yazılmasını göstərəcəm. Bu proqram müəyyən edəcək – klaviaturadan verilmiş ədəd sadədir yoxsa mürəkkəb. (Yadınıza salın ki, sadə ədəd yalnız özünə və 1-ə qalıqsız bölünür.)

Bu misalda da 2.4 misalında olduğu kimi həmin əsas məntiq istifadə olunur. Bu artıq hesab olunursa, onda məni bağışlayın.

Sadə ədədə yoxlanılması məntiqi:

	i dəyişənə 2 rəqəmini mənsub etmək.
	i dəyişəninin dəyəri n rəqəminin kvadrat kökündən kiçik və ya bərabər olana qədər,
	Əgər n ədədi i ədədinə qalıqsız bölünürsə, n ədədi sadə deyil.
	i dəyişənin dəyəri üzərinə 1 gəlmək.

for instruksiyasında da həmin məntiqdən istifadə olunur. Lakin for instruksiyasında i dəyişənin dəyərini bu instruksiyası özü artırır. Ona görə də, for dövründən istifadə etməklə proqramın kodu olduqca azalır:

2-dən n ədədinin kvadrat kökünə qədər bütün ədədlər üçün (for intruksiyası)
Əgər n ədədi i ədədinə qalıqsız bölünürsə, n ədədi sadə deyil.

Bu da proqramın tam kodu. Yadınıza salım ki, bu, 2.4-də çəkdiyimiz misalın digər versiyasıdır. Ona görə də, kodun çox hissəsi sizə artıq tanışdır.

Kod 2.1----------

```
#include <iostream>
#include <math.h>
using namespace std;

int main() {
	int n;			// yoxlanılan ədəd
	int i;			// dövrün artırıcısı
	int is_prime;		// bul bayrağı

	// Ədədin mürəkkəb olduğunu sübut etməyənə qədər, onun sadə olduğunu
	// nəzərə alaq

	is_prime = true;

	// Klaviaturadan rəqəm oxutmaq
	cout << “Enter a number and press ENTER: “;
	cin >> n;

	// Sadə ədədə yoxlanılması

	for (i = 2; i <= sqrt((double) n); i++) {
		if (n % i == 0)
			is_prime = false;
	}

	// Nəticələri çap etmək

	if (is_prime)
		cout << “Number is prime.”;
	else
		cout << “Number is not prime.”;

	return 0;
}
```

Proqram başladandan sonra əgər 23 ədəd daxil edilibsə proqram çap edəcək:

```
Number is prime.
```

Bu necə işləyir?

Proqramın başlığında *#include* direktivası istifadə olunur. Yəqin artıq bilirsiniz ki, bu C++ dilinin lazımi kitabxanalarında istifadə etmək üçündür. Bu proqramda kvadratın kökünü tapmaq üçün *sqrt* funksiyası istifadə olunur, ona görə də onu istifadə edə bilmək üçün *math* kitabxanası əlavə olunub.

```
#include <iostream>
#include <math.h>
```

Bundan sonra əsas *main* funksiya müəyyən olunub. Başlıca olaraq bu funksiya üç dəyişən elan edir, hansı ki, onlar lazımi yerdə istifadə olunur.

```
int n;          // yoxlanılan ədəd
int i;          // dövrün artırıcısı
int is_prime;   // bul bayrağı
```

is_prime dəyişəninin növü tamrəqəmli olduğuna baxmayaraq, biz onu **true** (1) və **false** (0) dəyərlərini yadda saxlamaq üçün istifadə etmişik. Diqqət yetirin ki, əgər sizin C++ dilinin versiyası *bool* növünü dəstəkləyirsə, sizin ondan istifadə etməyiniz gərəkdir:

```
bool is_prime; // bul bayrağı
```

Əgər proqram bölünəni tapa bilməsə, o ədədin sadə olduğunu çap edəcək. Ona görə də *is_prime* dəyişənin ilk dəyəri *true*-dir. Digər söz ilə desək, bu dəyişənin dəyəri *false* edilməsə, bu, ədədin sadə olduğunu göstərəcək.

```
// Ədədin mürəkkəb olduğunu sübut etməyənə qədər, onun sadə olduğunu
// nəzərə alaq
is_prime = true;
```

Proqramın ürəyi *for* instruksiyasıdır, hansı ki ədədin sadə olub-olmadığını yoxlayır. 2-ci başlıqda başa saldığım kimi, bölünənlərini n ədədinin kvadrat kökünə qədər yoxlamaq zərurridir. Əgər bu ədədlər arasında bölünən tapılmadısa, deməli yoxlanılan ədədin özündən başqa bölünənləri yoxdur.

n % i ifadəsinda modula görə bölmək operatoru istifadə olunub, hansı ki, qismətdən qalığı götürmək üçün. *n* ədədi *i* ədədinə qalıqsız bölünürsə, qalıq 0-a bərabərdir, - bu halda *n* ədədi sadə deyil.

```
for (i = 2; i <= sqrt((double) n); i++) {
        if (n % i == 0)
                is_prime = false;
}
```

for operatorun necə işləməsi haqqında yadda saxlayın: birinci ifadə - *başladan*, ikinci – *şərt*, üçüncü – *artırıcı* adlanır. Bu *for* dövrü aşağıdakı aşağıdakı koda ekvivalentdir:

```
i = 2;
while ( i <= sqrt((double) n)) {
        if (n % i == 0)
                is_prime = false;
        i++;
}
```

Diqqət yetirin ki, *for* dövrü olan versiyada onun daxilində yalnız bir instruksiya istifadə olunub – bu *if* instruksiyasıdır.

Məsələlər

Məsələ 3.3.3. *Misalda elə dəyişikliklər edin ki, o sərfəli koddan istifadə etsin. Müasir mikroprosessorlara diqqət yetirərək siz fərqi hiss etmərsiniz, lakin artıq böyük ədədlərlə işi zamanı (milyonlar, milyardlarla) fərq göz qabağında olacaq.(Onu nəzərə alın ki, böyük ədələr arasında sadə ədədə çox nadir hallarda rast gəlmək olar.)*

Lakin yenə də, aşağıdakı dəyişikliklər kodu daha sürətli edəcək.

- ✓ ***square_root_of_n*** *dəyişəni elan etməklə* ***n*** *ədədin kvadrat kökünü yalnız bir dəfə müəyyən edin. Onu dəyərini for dövrəsinə girişdə müəyyən edin. Bu dəyişənin növü* ***double*** *olmalıdır.*
- ✓ ***n*** *ədədinin bölünəni tapılan zaman axtarışı davam etməyin mənası yoxdur. Ona görə də, axtarışı sona yetirmək üçün* ***if*** *instruksiyasından sonra* ***break*** *operatorunu (dövrü sona yetirmək üçün) yazın.Əlbəttə ki,* ***is_prime*** *dəyişənin dəyərini* ***false*** *edəndən sonra.*