AZƏRBAYCAN RESPUBLIKASI TƏHSIL NAZIRLIYI
SUMQAYIT DÖVLƏT UNİVERSİTETİNİN NƏZDİNDƏ
SUMQAYIT DÖVLƏT TEXNIKI KOLLECI

“Müasir proqramlaşdırma dilləri”

Orta ixtisas Təhsil müəssisələrində fənnin tədrisi üçün

DƏRS VƏSAİTİ

SUMQAYIT 2020

Mövzuların adları

1. Proqram sistemleri haqqında
2. C/C++ proqramlaşdırma dilinin qısa tarixi.
3. C++ dilində yazılmış proqramların icra olunma mərhələləri.
4. C++ dilinin identifikatorlar, açar sözlər.
5. Sabitlər, işarələr.
6. Verilənlərin tipləri.
7. Ifadələr.
8. Proqramın strukturu.
9. Dəyişənlərin daxili və xarici səviyyədə elanı.
10. Sadə hesabi çevrilmələr.
11. Multiplikativ əməliyyatlar.
12. İnkrement və Dekrement.
13. Münasibət əməliyyatları, mərtəbə məntiqi əməliyyat, məntiqi əməliyyatlar.
14. Şərti kecid operatoru.
15. Giriş-çıxışın təşkili.
16. Dövri alqoritmlərin proqramlaşdırılması.
17. Son şərtli dövr operatoru.
18. Continue operatoru, return operatoru.
19. Proqramların layihələndirilməsi.
20. Massivlərin elanı, birölçülü massivlərinə müraciət,
21. Dinamik və adi dəyişənlər.
22. Simvollar və sətirlər.
23. Funksiyalar.

Proqram sistemleri haqqında

Proqranılaşdırma sistenıləri proqramlaşdırma dillərində işləməyi təmin edirlər. Buraya proqram- laşdırma dilləri, həmin dillərdə proqraınları kompüter dilinə çevirən translyatorlar (çevirici proqramlar), sazlayıcı proqramlar və s. daxildir.
Kompüter dili bılavasitə kompüterin "başa düşdüyü" kodlarda ifadə olunmuş əmrlərdən təşkil olunur. Bu halda proqram müəyyən əmrlər ardıcılltğından ibarət olur. Bu əmrlər kifayət qədər sadə olub, verilәıılər üzərində müəyyən əməliyyatları (toplama, çıxma, vurma, bölmə, ınüqayisə, köçürmə və s.) yerііıə yetirirlər. Hər bir əmr yerinə yetirilən əməliyyat (əməliyyatın kodu), əməliyyatda iştirak edən operandlar (verilənlərin yaddaşdakı ünvanlan və ya özləri) və nəticənin haraya (hansı iinvana) yazılması haqqında ınəlumatdan ibarət olur. Hər bir tip kompüter üçün müxtəlif əmrlərin sayı 100-dən artlq olur.

Kompüter dilləri kompüterin tipindon asılı olaraq müxtəlif olduqlarına görə, istitadəçilər üçün öyrənilməsi çətin və işlədilməsi çox zəhmət ləlәb etdiyindən, əlverişli deyillər. Ona görə də təbii dila yaxın formallaşdınltnış dillərdən istifadə olunur. Bu cür dillərə proqramlaşdırma dilləri deyilir. Bəzən bu mənada "yüksək səviyyəli proqramlaşdırma dilləri" ifadəsindən də istifadə olunur. Proqramlaşdırma dillərində yazılmış proqram (ona **ilkin proqram** deyilir) sonradan koınpüter dilinə çevrilir, sazlamr və icıa olunur. Kompüter dilindəki proqrama **işçi** və ya **mütləq proqram** deyilir. İlkin proqramı işçi proqrama çevirmək üçün **translyator** adlanan xüsusi proqramlardan istifadə olunur.

İstifadə olunan dilin strukturuna, formallaşdırma səviyyəsinə və vəzifəsinə uyğun olaraq proqramlaşdırma sistemlərini aşağıdakı siniflərə bölmək olar:

- maşınyönlü sistemlər;
- proseduryönlü sistemlər
- problemyönlü sistemlər;
- köməkçi sistemlər.

Maşınyönlü sistemlərdə proqramlaşdırma dili müəyyən kompüterlə və ya kompüter ailəsi ilə əlaqəli olur. Bu sislemlәriıı tipik nümayəndələri simvolik proqramlaşdtrma sistemləri, avtokodlar, makrogeneratorlar və assemblerlərdir.

Hazırda assemblerlər geniş tətbiq olunur. Bu sistemlərdə istifadə olunan assembler dili makroəmıiərdən təşkil olunur. Makroəmr müəyyən əməliyyatı və ya funksiyanı yerinə yetirmək üçün bir və ya bir ııeçə maşın əmrindən ibarət olur. Hər bir kompüter ailəsinin özünə məxsus assembler dili mövcuddur. Assembler dilində işləmək nisbətən çətin olur, çox vaxt aparır. Lakin bu dildə yazılan proqram digər dillərə nisbətən daha yığcam olduğundan, və onların icra vaxtı nisbətən az olduğundan, istehsalat sahələrində böyük tezliklə həll oluan məsələlərin assenıblerdə proqramlaşdırılması məqsədəuyğundur. Praktikada assembler dilindən həm bu məqsədlə, həm də sistem proqramlaşdırılmasında geniş isitifadə oiunur. Assembler dilindəki proqramı kompüter dilinə çevirən proqram "Assembler" adlamr.

Proseduryönlü sistemlərdə istif'adə olunan proqramlaşdırma dilləri maşınyönlü dillordən fərqli olaraq, konkret tip kompüterlə əlaqəli olmayıb, istənilən alqoritmlərin (prosedurlarm) proqramlaşdırmasım və bu proqramlarm istənilən tip kompüterdə icrasını təmin edirlər. Bu dillərin iiç adı mövcuddur: alqoritmik dillər, prosedur dilləri, direktiv dillər. Oıılara yüksək səviyyəli proqramlaşdırma dilləri də deyilir.

Yiiksək səviyyəli proqramlaşdıma dilləri universal xarakter daşıyıb.
istənilən sahəyə aid ınəsələlərin proqramlaşdırılmasını təmin edirlər. Lakin proqramlaşdırma təcrübəsində çox vaxt proseduryönlü dil tətbiq sahəsinin xarakterinə uyğun yaradılır. Bu baxımdan proseduryönlü dilləri şərti olaraq 4 qrupa ayırmaq olar:

- elmi-texniki ıııəsələlərin proqramlaşdırılması üçün dillər. Bu qrupa Alqol, Fortran, Basic, Pascal, C dillərini aid etmək olar;
- - iqtisadi məsələlərin proqramlaşdırılması üçün dillər: Kobol, PL-1;
- texnoloji proseslərin idarəetmə alqoritmlərinin və modelləşdirmə məsələlərinin proqramlaşdınlması üçün dillər: ART, Simula, Simskript;
- informasiya-məntiq məsələlərinin həlli üçün dillər: LİSP, Komit, FPL, KRL.

Problenıyönlii sistemlərdə həll olunan məsələnin alqoritmini qunnağa ehtiyac olmur. Bu sislemlər dar çərçivədə eyni tipli məsələlrin hallinə yönəldilir. Problemyönlü dillərə misal olaraq mühəndis məsələlərinin həlli üçün yaradılan xüsusi diləri (ART, ADART, SYMAP, APROKS), ekspert sistemlərinin yaradılması üçün istifadə olunan PROLOG dilini göstərmək olar. PROLOG dilində məntiqi çıxarış mexanizminin qurulması və idarə olunması verilənlərə əsaslanır. Bu sistemlərə

homçinin hesabatlar generatorları (məsələn, RPQ), çeşidləmələr generatorlan, cədvəl generatorları (məs., EXCEL) aiddir.

Kömokçi sistcnılər verilәıılərin emalı zamanı bir sıra köməkçi funksiyaları yerinə yctinnək üçün əvvəlcədən hazırlanmış proqramlar toplusundan ibarət olur. Köməkçi sistemlərin komponentləri, məsələn, sazlayıcı proqramlar. proqramlaşdırma sistemləri ilə birlikdə istifadə olunur. Sazlayıcı proqram işçi proqramı yoxlayıb, səhviəri aşkar edir.

Yuxarıda qeyd etdiyimiz kimi, kompüter yalnız maşm dilində işlədiyi üçün proqramlaşdırma dilində yazılan proqramı maşm dilinə çevirmək lazımdır. Bu işi **translyator** adlanan proqramlar kompleksi yerinə yetirir. Funksional təyinatından asılı olaraq translyator 3 cür ola bilər: interpretator, kompilyator, assembler. Onlar arasında fərq çcvrilən proqrannn mətninin müxtəlif üsulla emal olunmasındadır.

İnterpretator ilkin proqramın cümlələrini (operatorlarım) bir-bir təhlil edib. kompüter dilinə çevirir və icra edir. Növbəti operatorun emalından sonra o birisinə kcçilir. Sonuncu operatorun emalından sonra interpretasiya prosesi və proqramm icrası başa çatır. İnterpretasiya üsulu ilə proqramın kompüter dilinə çevrilməsi və icıası ləng gedir. Bu onunla əlaqədardır ki, məsələn. dövri prosesin icrası dövrə daxil olan operatorlann dövrləriıı sayı qədər təhlilini və çevrilməsini tələb edir. Odur ki, translyasiyanın bu üsulu səmərəli deyil. Lakin inteıpretator proqraımn sazlanması üçün əlverişlidir. İnterpretator proqramı istənilən operatordan başlayaraq emal etməyə və proqıamın icrası zamanı dəyişənlərin aidıqları qiymətləri yoxlamağa imkan verir. Dialoq rejimiııdə proqramda istənilən düzəlişlər aparmaq və proqramı təkrarən icra etınək mümkündiir.

Kompilyator, interpretatordan fərqli olaraq. ilkin proqraını bütövlükdə rnaşın dilinə çevirir. Proqramda morfoloji və sintaksis səhviər olarsa, onları aşkar edib, istifadəçiyə xəbər verir. Səhviər düzəldildikdən sonra kompilyasiya yenidən davam etdirilə bilər. ya da saxlanmaq üçün xarici yaddaşa köçürülə bilər.

İlkin proqramın operatorlarının təhlili və çevrilınəsi bir dəfə aparıldığı üçün kompiiyatorun sürəti yüksək olur. İşçi proqraının icrası kompliyasiya prosesindən asılı olmadığı üçün, proqramın icrası zamanı kompilyatoaın ƏYQ-də olmasına ehtiyac olmur.

Beləliklə, aşağıdakı nəticə çıxarıla bilər: proqramın sazlanması zamanı interpretatordan istifadə etmək, sazlanmış proqramı isə koınpilyator vasitəsilə emal etınək məqsədəuyğundur.

Assembler - assembler dilindəki proqramı kompüter dilinə çevirən proqramdır. Assembler ışçi proqramı bir gedişlə və ya çox gedişlə maşın diJinə çevirə bilər. Daha səmərəli işçi proqram çoxgedişli asscmblerlərdən istifadə etməklə alınır.

Texniki xidmət proqramları kompüterin diizgün işləməsinə nəzarət etmək və nasazlıqlan aşkar etmək üçündür. Kompüterin işinə nəzarət etmək üçün müxtəlif iisullar mövcuddur. Bu üsullardan bəziləri kompüterin aparat vasitoləri ilə, bəziləri aparat-proqram vasitələri ilə, bəziləri isə proqram vasitələri ilə həyata keçirilir.

Proqramla nəzarət test proqramları və xüsusi nəzarət proqraınları vasitəsilə həyata keçirilir. Testlə yoxlama kompüterin və onun ayrı-ayrı bloklarımn işini yoxlayan test-proqramlar vasitəsilə yerinə yetirilir.

Test proqramları adətən kompüterin daimi yaddaş qurğusunda saxlanıhr və kompüter elektrik şəbəkəsinə qoşulduqda avtomatik olaraq işə düşürlər.

Xüsusi nəzarət proqramları kompüterdə məsələlərin həlli üçün tətbiq olunan proqramların icrası zaınam əvvəlcədən müəyyənləşdirilmiş vəziyyətlərin,. asıhlıqlarm və məhdudiyyətlərin ödəniiib-ödənilməməsini yoxlayır. Bütün hallarda nasaziıqlarıı xarakteri, mənbəyi və bəzən də səbəbi haqqında ekrana və ya çapa məlumat xaric edilir.

Proqramlaşdırma dilləri

Perl 80-ci illərdə Larri Uoll tərəfindən işlənmişdir. Bu proqram dilinin böyük həcmli mətn fayllarının effektiv işlənməsində, hesabatların generasiyasında və məsələlərin idarəsində istifadəsi nəzərdə tutulmuşdur.

Perl-dən sətirlərlə, massivlərlə, ayrı-ayrı verilənlərlə, proseslərin idarəsində, system informasiyaları ilə işlərdə istifadə edilir. HTML web səhifələrin hazırlanmasında istifadə edilən populyar dildir.

Assembler-dən başqa, qalan proqramlaşdırma dillərinin hər biri yüksək səviyyəli dil

adlandırılır, ancaq bu hələ onların eyni səviyyəli olması demək deyildir. Bir dilin səviyyəsi başqasının səviyyəsindən yuxarı, aşağı ola bilər.
"Yüksək səviyyəli dil" dedikdə, onun insan dilinə nə qədər yaxın olması başa düşülür. Beləliklə, insan üçün daha anlaşıqlı olan və proqramlaşdırma prosesini asanlaşdıran yeni dillər yaradılmağa başladı.
Proqramlaşdırma dilinin hər bir yaradıcısı insan-maşın əlaqələri haqqında öz təsəvvürlərini gerçəkləşdirdiyindən, qısa müddət ərzində yüzlərlə yeni dil meydana çıxdı. Yüksək səviyyəli dillər adlandırılan dillərin az bir qismi öz yerini tapıb inkişaf etdi və möhkəmləndi.
Yüksək səviyyəli dillərin öz müsbət cəhətləri var. Yüksək səviyyəli dilin Assembler-dən başlıca üstünlüyü, onu öyrənmək və istifadə etməyin çox-çox asan olmasıdır. Yüksək səviyyəli dildə yazılmış proqram Assembler-dəkinə nisbətən, daha yığcam və anlaşıqlıdır.
Onlar, əsasən, daşınabiləndir, yəni müxtəlif prosessorlu kompüterlərdə eyni cür işləyir. Bu isə o deməkdir ki, proqramı yazarkən onun işləyəcəyi kompüterin arxitekturasının incəliklərini öyrənməyə ehtiyac qalmır.
Əlbəttə, bu halda hər bir prosessorun öz kompilyatoru olmalıdır və onun yaratdığı icra faylı yalnız həmin prosessor üçün yararlı olacaqdır.

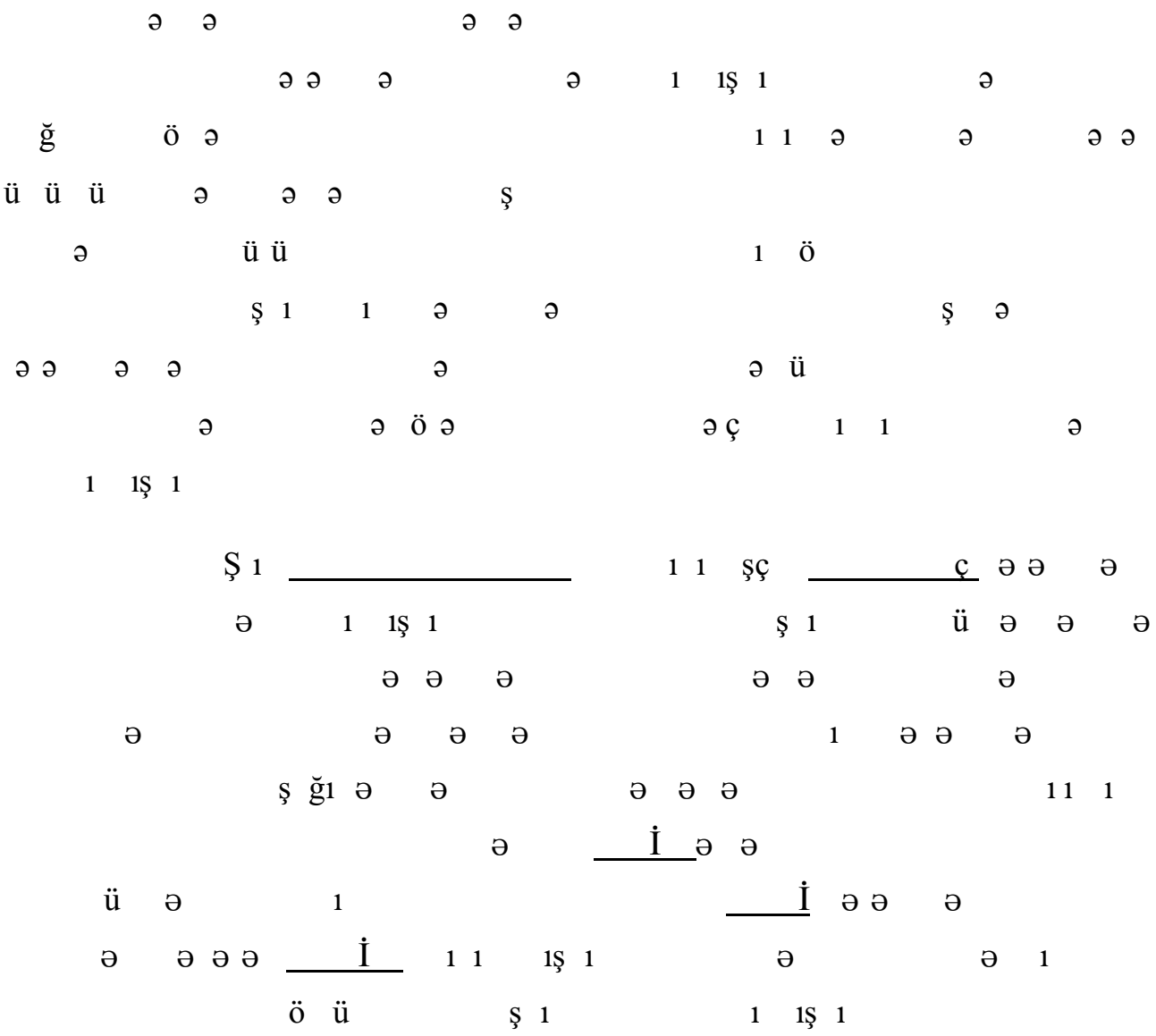
Proqramlaşdırma dillərinin müxtəlif səviyyələri var. Əsasən 5 qrupa ayrılırlar:
1. Çox yüksək səviyyəli dillər və ya vizual dillər: Access, FoxPro, Paradox, XBase, Visual Basic.
2. Yüksək səviyyəli dillər (bunlara bəzən "alqoritmik dillər" də deyilir): Pascal, Basic, Fortran
3. Orta səviyyəli proqramlaşdırma dilləri: C, C++
4. Aşağı səviyyəli proqramlaşdırma dilləri: Assembly language
5. Maşın dili: Ən aşağı səviyyəli dil olub, 0 və 1-lərdən ibarətdir.

Bundan başqa, proqramlaşdırma dillərini 2 ayrı qrupa da bölmək olar:
1. Prosedur proqramlaşdırma dilləri (Pascal, Basic)
2. Obyekt yönümlü proqramlaşdırma dilləri (C++, Java, Smaltalk)

Hər hansı proqramlaşdırma dilini istifadə etmək üçün isə, bizə ilk növbədə kompilyator lazımdır. Kompilyator olduqdan sonra, biz öz proqramımızı mətn redaktorunda (məsələn, Notepad) yaza bilərik.

Lakin bizə daha çox IDE, yəni proqramlaşdırmanın inteqrallaşmış mühitindən istifadə edirik. Sadə dillə desək, bizə bir mühit verilir, orda komponentlər olur və bunlardan istifadə edərək, biz öz proqramımızı yazırıq.
Və ən əsası, proqramlaşdırmanı öyrənmənin yolu proqram yazmaqdır.

C++ dilində yazılmış proqramların icra olunma mərhələləri



C++ dilində yazılan sadə bir proqramı nəzərdən keçirək:

Misal. ”Azərbaycan Texniki Universiteti” sözünü displeyin ekraninda əks etdirən proqram tərtib edək.

```
// AzTU.CPP- proqram aylının adı
# include <iostream.h>
void main ()
{
   cout <<”\n Azərbaycan Texniki Universiteti”  ;
}
```

Proqramın birinci sətri ‘//’ simvolu ilə başlayıb, görünməyən “sətrin sonu” simvolu ilə qurtaran birsətrli şərhi göstərir.

Proqramın ikinci sətrində preprosessorun # include <iostream.h> direktivi yerləşir. Direktiv kompilyatora proqram mətninə iostream.h faylının tərkibini əlavə etmək məlumatını verir. Bu direktiv verilənlərin standart axınla daxiledilməsini və xaricediləməsini təmin edir. Göstərilən vasitələr iostream.h adlı faylda yerləşir. Burada “i” (input) – daxiletmə, “o” (output) – xaricetmə, stream – axın, “h” (head) – başlıq deməkdir. Susmaya görə standart axınlı xaricetmə displyin ekranına xaric etməni, standart axınlı daxiletmə isə verilənlərin klaviaturadan daxil edilməsini təmin edir.

Proqramın növbəti hissələri isə baş funksiyanın təsviridir. Bu baş funksiyanın başlanğıcı ilə başlayır:

```
void main ( )
```

C++ dilində istənilən proqram yalnız bir main adlı baş funksiyadan təşkil olunur. Proqramın icrası buradan başlayır. void spesifikator adlanır və main funksiyanın yerinə yetirilməsi nəticəsində heç bir qiymət qaytarılmadığını göstərir. main – dən sonra mötərizədə parametrlərin siyahısı da ola bilər. Bu misalda parametrlər lazım deyil və siyahısı boşdur.

İxtiyari funksiyanın gövdəsi fiqurlu mötərizədə olan təsvir, təyin və operatorlardan ibarətdir. Hər bir təsvir, təyin və ya operator “ ; ” simvolu ilə qurtarır. Burada baş funksiyanın gövdəsində təsvir, təyin yoxdur və yalnız informasiyanı displeyin ekranına xaric etmək üçün cout<<”\n Azərbaycan Texniki Universiteti”; operatoru var. İnformasiya xaric olunmaq üçün <<

əməliyyatı vasitəsi ilə cout obyektinə ötürülür. Bu halda bu ”\n Azərbaycan Texniki Universiteti”; sətridir (sətr sabiti). C++ dilində dırnaq işarəsi daxilində olan ixtiyarı simvollar ardıcıllığıdır. Bu simvollar içərisində ekranda təsvir olunmayan idarəedici simvollar da ola bilər. Məsələn, burada ‘\n’- idarəni ekranın növbəti sətrinin başlanğıcına ötürən idarəedici simvoldur. Qeyd edək ki, binar əməliyyatlarda << simvollar cütü sola sürüşdürmədir.

C++ dilində icra olunan proqramın hazırlanma sxemi aşağıdakı şəkildə göstərilib.

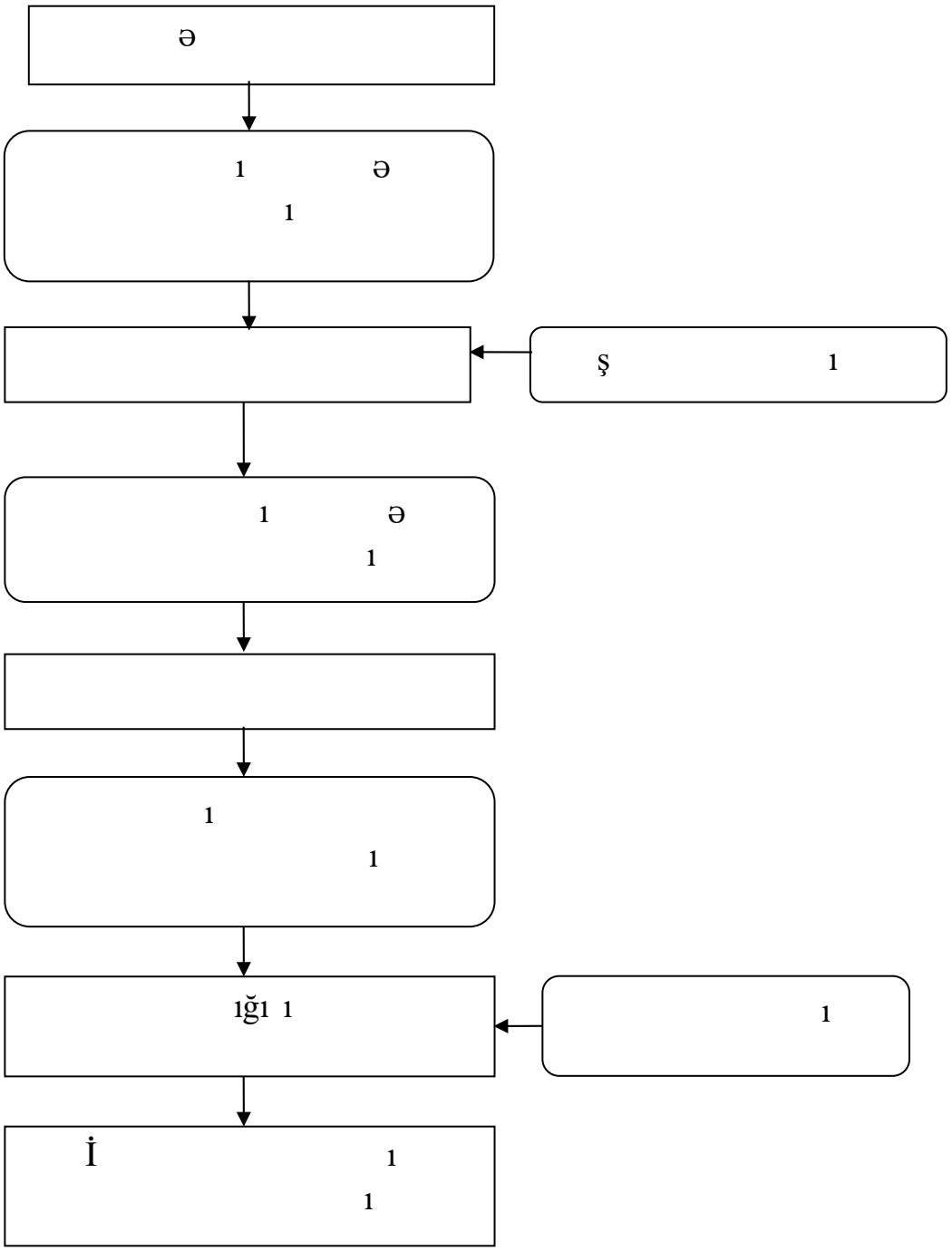
Bu sxemə uyğun olaraq proqramın yerinə yetirilməsi aşağıdakı mərhələrdən ibarətdir:

1. Mətn redaktorunun köməyi ilə proqramın mətni tərtib olunur və genişlənməsi cpp olan faylda saxlanır. Məsələn, bu faylın adı example.cpp – dir.

2. Funksiya başlığından əvvəl yerləşən preprosessor direktivi ilə preprosessorun emal prosesi həyata keçirilir. Məsələn, # include direktivi ilə preprosessor standart kitabxanadan *.h fayllarını proqram mətninə qoşur.

3. Bu mərhələdə proqram mətninin kompilyasiyası baş verir. kompilyasiyada sintaksis səhvlər aradan qaldırılır və genişlənməsi obj olan faylda proqramın obyekt faylı alınır. Məsələn, example. obj.

4. Yığıcı (Linker) sistem proqramın köməyi ilə obyekt fayla kitabxananın lazım olan faylları əlavə olunaraq genişlənməsi exe olan faylda icra olunan proqram yaradılır. Məsələn, example.exe


ə
1 ə
1
ş 1
1 ə
1
1
1
ığı 1
1
İ 1
1

C++ dilinin əlifbası, identifikatorlar, açar sözlər

ə 1

Hər bir dildə olduğu kimi, C++- un öz qramatik qaydaları vardır ki, əgər proqramın yazılışı zamanı bu qaydalara əməl edilməzsə kompilyator səhvləri aşkar edərək işi dayandıracaq. C++- un əlifbasına aşağıdakılar daxildir.

1. Açar sözlərin yazılışı üçün latın əlifbasının kiçik simvollarından digər sözlərin yazılışında böyük simvollarından da istifadə edillə bilər. (A-Z, a-z)
2. Ədədlərin təsviri üçün 0,1, …, 9 rəqəmlərindən istifadə edillə bilər.
3. Xüsusi işarələrdən:

   " { } | [ ] ( ) + - / % \ ; ' . : ? < = > _ ! & * # ~ ^, istifadə edillə bilər.

4. Görünməyən simvollar aralıq(boşluq) , TAB , ↵

Proqramda izahatlar vermək üçün şərhlərdən(açıqlamalardan) istiadə edilir. Açıqlamalar // simvollarindan sonra gələn simvollar ardıcıllığıdır.

Məsələn:

// AZTU AHT FAKULTESİ

C++ dilində altı sinif leksem var: sərbəst seçilərək istifadə edilən identifikatorlar, xidmət edici açar sözlər, sabitlər, işarə sətri, əməliyyat işarələri və ayırıcılar. İndi isə dilin leksik elementlərinə ətraflı baxaq.

İ

" - ” və ya latın hərfi ilə başlayan 31-ə qədər latın hərfləri və ya rəqəmlər ardıcıllığına identifikator deyilir. Məsələn:

A _1, _ BİT, _ Bit, _ bit

Böyük və kiçik simvolların bir-birindən fərqləndiyinə görə axırıncı üç identifikator müxtəlif obyektlərin adlarıdır. Müxtəlif sözləri bir identifikatorda birləşdirmək üçün boşluq simvolu yerinə _(alt xətt) simvolundan istifadə edilir. Məsələn : maksimum_element.

ç ö ə

C++ dili tərəfindən ehtiyata alınmış və proqramçı tərəfindən sərbəst ad kimi seçilməyən sözlərə açar sözlər deyilir. Açar sözlər verilənlərin tipini, yaddaş sinfini, modifikatorları, psevdo dəyişənləri və operatorları tə’yin edir və onlar aşağıdakılardır.

asm	delete	if	register	throw
auto	do	inline	return	try
break	double	int	short	tupedef
case	else	long	signed	typeid
cath	enum	new	sizeof	union
char	extern	oporator	static	unsigned
class	float	overload	struct	virtual
const	for	private	switch	void
continue	friend	protected	template	volatile
default	goto	public	this	while

Verilənlərin tipinin tə'yinində aşağıdakı spesifikator və kvalifikatorlardan istifadə edilir:

char –işarə

double – sürüşkən nöqtəli həqiqi ikiqat uzunluqlu

enum –sadlanan , tam sabitlərin sadalanması

float – həqiqi sürüşkən nöqtəli

int – tam

long – uzun tam

short – qısa tam

struct – struktura

signed – işarəli tam

union – birləşmə

unsigned – işarəsiz tam

void – qiymətin qaytarılmaması

typedef – tipə uyğun sinonimin tə'yini.

Kvalifikatorlar kompilyatora proqram kodunun optimallanması zamanı obyektlərin xüsusi e'mala malik olması informasiyasını verir və onlar aşağıdakılardır:

const – obyektin yalnız oxuna bilinməsini, yə'ni sabit olmasını göstərir.

volatile – proqramçının göstərişi olmadan da obyekt qiymətini dəyişə bilər (əməliyyat sistemi tərəindən).

Yadaş siniflərin tə'yini üçün

auto – avtomatik

extern – xarici

register – registrli

static – statik

açar sözlərindən istifadə edilir.

Operatorların təşkili üçün

break – dövrdən çıxış

continue – dövrü davam etməli

do – dövrü yerinə yetirir

for – üçün (dövr)

goto – şərtsiz keçid

if – əgər şərtli keçid

return – funksiyadan qayıdış

switch - çevirici

while – nə qədər ki (dövr)

Xidmətçi açar sözlərə aşağıdakılar aiddir:

default – switch operatorunda variant olmadıqda istifadə edilir;

case – switch operatorunda variantı tə'yin edir;

else – əks halda (if - də);

sizeof – operatorun uzunluğunu tə'yin edir.

Modifikatorlar aşağıdakılardır:

asm, cdecl, _ cs, _ ds, _ es, _export, _loadds, _regparam, _ ss, _saveregs, _seg, far, fortran, huge, interrupt, near, pascal.

Register dəyişənləri üçün aşağıdakı xidməti sözlər daxil edilmişdir:

_AH	_BH	_CH	_DH	_DI	_SP	_SS
_AL	_BL	_CL	_DL	_SI	_CS	_ES
_AX	_BX	_CX	_DX	_BP	_DS	_FLAGS

Bu açar sözləri saymasaq da olardı. Lakin proqramçı bilməlidir ki, bu sözlərdən o, sərbəst istifadə edə bilməz. Proqramçı dəyişənlərin adlarını _ (alt xətt) və ya _ _(iki alt xətt) ilə başlaması məsləhət deyil. Çünki belə adlı obyektlər kompilyatorun proqramxanasında da ola bilər. Ümumiyyətlə, dəyişənlərin adlarını böyük simvollarla başlamaq məsləhətdir.

Sabitlər və işarələr

Proqramda qiymətlərini dəyişməyən kəmiyyətlərə sabit deyilir. C++- da 5 tip sabit vardır: simvol, sadalanan, tam, həqiqi tip sabitlər və NULL göstərici. NULL - dan başqa qalan sabitlərə hesabi sabitlər deyilir. Tək dırnaq arasına alınan sabitə simvol (işarə) tip sabit deyilir.

Məsələn: 'A ' , ' 9 ' , '-'.

Tək dırnaq arasında printerdə və ya displeydə əks olunan istənilən sabiti saxlamaq olar. Lakin tək dırnaq arasında \ -ilə başlayan simvollar ilə printerdə və ya displeydə görünməyən əməliyyatları da apara bilərik. Bunlara eskeyp ardıcıllıq deyilir və onlar aşağıdakılardır:

Təsviri	Kodu	İşarə edilən simvol	Yetirdiyi iş
\a	0x07	bel(audible bell)	Səs siqnalı
\b	0x08	bs(backspace)	Bir addım geri
\f	0x0C	ff(form feed)	Səhifəyə keçid
\n	0x0A	İf(line feed)	Sətrə keçid
\r	0x0D	cr(carriage return)	Karetkanın qayıtması
\t	0x09	ht(horizontal tab)	Üfüqi tabulyasiya
\v	0x0B	vt(vertikal tab)	Şaquli tabulyasiya
\\	0x5C	\ (backslash)	Əks sleş
\'	0x27	(single quote)	Apastrof
\"	0x22	" (double quote)	Dırnaq
\?	0x3F	?(question mark)	Sual işarəsi

Bu simvollar eskeyp ardıcıllığın bir hissəsidir. Bu leksemlər '\ddd', '\xhh' və ya '\Xhh' kimi də tə'yin edilə bilər.

'\ddd' istənilən simvolun 8 – lik say sistemindəki yazılışıdır.

Məsələn:

'\017' , '\0155'

'\xhh' və ya '\Xhh' istənilən simvolun 16 –lıq say sistemindəki yazılışıdır.

Məsələn:

'\x05' , '\X1C'

Aşağıdakı proqramı nəzərdən keçirək.

```
// Simvol sabitlərin eskeyp ardıcıllığ kimi yazılışından istiadə
# include <iostream.h>
```

```
void main()
{
cout<<'\x0A'<<'\x53'<<'\x61'<<'\x6c'<<'\x61'<<'\x6d'<<'\x2c';
cout <<'\040'<<'\126'<<'\141'<<'\154'<<'\145'<<'\148'<<'\041';
}
```

Bu proqramın icrasından sonra ekranda

Salam, Valeh! əks olunacaq.

Simvol sabiti tam tip sabit kimi istifadə etmək olar.

ə

C++ dilində tam sabitləri onluq, səkkizlik və onaltılıq sabitlərə bölə bilərik. Onluq tam sabitlər sıfır ilə başlamayan onluq rəqəmlər ardıcıllığından ibarətdir.

Məsələn:

44, 659,0

Sıfır ilə başlayan 7 rəqəmindən böyük olmayan rəqəmlər ardıcıllığıdan ibarət ədədə səkkizlik tam sabit deyilir. Məsələn:

067,-0123

0x və ya 0X-ilə başlayan 16-lıq rəqəmlər ardıcıllığına 16-lıq tam sabit deyilir. Məsələn:

0x1A , 0XB26

Həqiqi sabitlər

[±] [T1 T2 …Tn] [.] [k1 k 2 … km] [E ±M1 M 2 M 3] şəklində olan sabitlərdir.

Məsələn:

533, 3.14, - 2.7E+5, .7E – 3

Verilənin yazılışında boşluqdan istifadə etmək olmaz. Məsələn: 2.7 E – 3 yazılışı düzgün deyil.

İşarələr sərti cüt dırnaq arasında yazılan istənilən işarələrdən təşkil edilir. Məsələn:

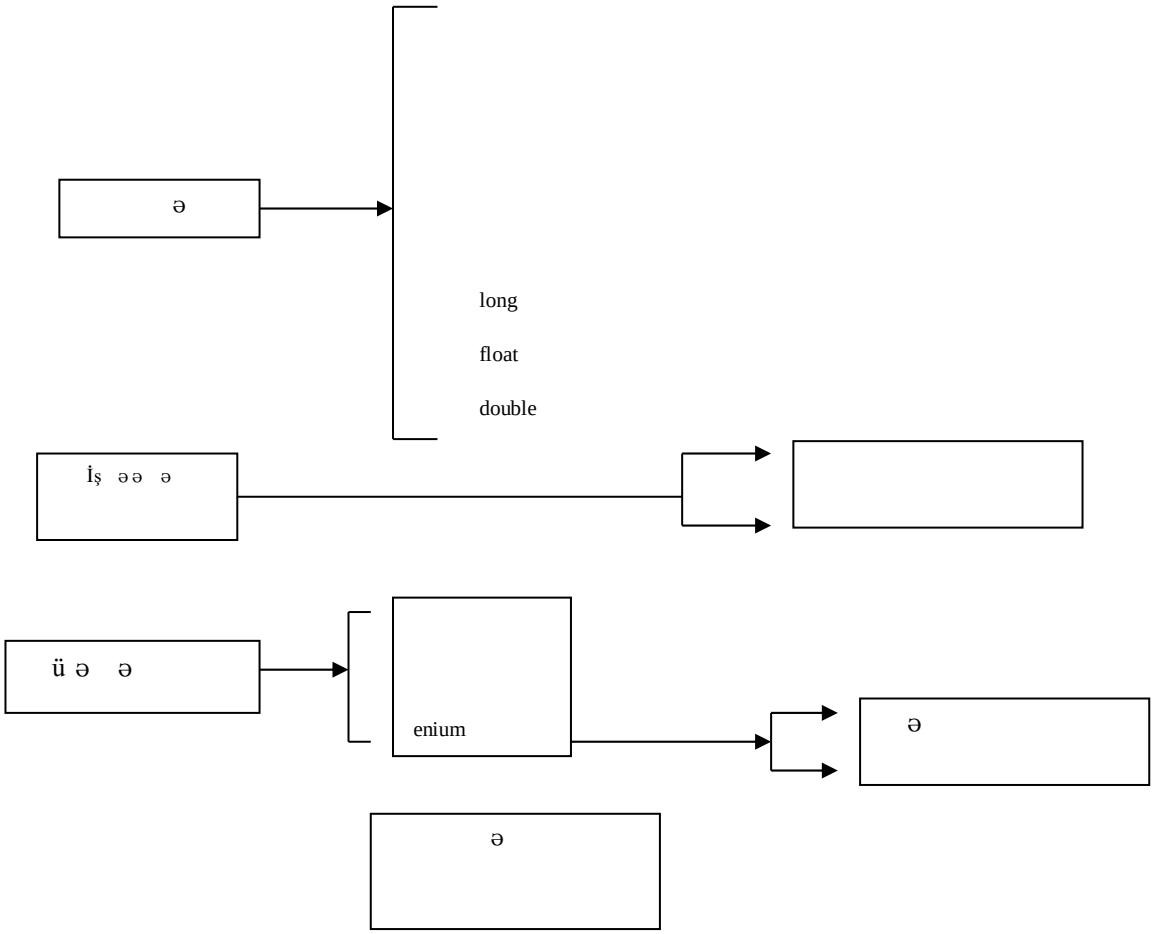
"GR 682A1 " , " AZERİ "

Dəyişən, dəyişənlərin elan qaydası

Proqramçının kompüterlə alış-verişi sahə deyilən yaddaş parçaları ardıcıllığı ilə edilir. Sahə içərisi eyni tipli verilənlərlə doldurulacaq bir qutuya bənzəyir. Bu sahələrin içərisi daima dəyişdirilə bildiyindən onlara dəyişənlər deyilir. Sahələr identifikatorlar ilə adlandırılır. Sahə adları aşağıdakı kimi tə'yin edilir.

[yaddaş sinfinin snesifikatoru] [sahə tipi] sahə adı [= sabit]

Yaddaş sinfi auto, extern, register və static sözləri ilə tə'yin edirlir. Yaddaş sinfi dəyişənlərə proqramın hansı hissəsindən müraciət olunacağını tə'yin edir.



Sahə tipləri aşağıdakılardır.

şə

ə ə ə

x , SUM , ATA _adı (düzgün yazılmış identifikatorlar)

37X const, x, 1 (səhv yazılmış identifikatorlar)

Tam saylı dəyişənlər aşağıdakı qiymətləri ala bilər.

Tip	Dəyişmə intervalı	tutduğu sahə
Unsigned char	[0,+255]	1b

Char	[-128,+127]	1b
İnt	[-32768,+32767]	2b
Short int	[-32768,+32767]	2b
Unsigned int	[0,+65535]	2b
Long int	[-2147483648,+2147483647]	4b
Unsigned long	[0,+4294967295]	4b

Göründüyü kimi tam saylı dəyişənlər işarəli və ya işarəsiz ola bilərlər. Məntiqi əməliyyatlarda işarə mərtəbələrinin iştirakının müəyyən mə'nası var. short və long üçün "int" sözünü yazmaya bilərik.

char – bir simvoldan ibarət veriləni tə'yin edir və 8 bitlik yer tutur. Simvollar ASCİİ kodlar cədvəli ilə tə'yin edilir. Məsələn:

```
int      a , b , c = 5 , t ;
char   Q, n = ' k ‘, L;
unsigned char A1, A2;
```

Həqiqi dəyişənlər aşağıdakı qiymətləri ala bilər.

Tip	Dəyişmə intervalı	Tutduğu sahə
float	± 1.7 E± 38	4b
double	± 1,7 E± 308	8b
long double	± 1, 7 E± 4932	10b

Həqiqi dəyişənlərin işləməsi üçün "Math.lib" proqramxanası olmalıdır və bu proqramxana avtomatik olaraq qoşulur. const tipi ilə tə'yin olunan dəyişənin qiyməti proqramda dəyişdirilə bilməz, yəni sabit qalır (yalnız pointer-lə dəyişdirilə bilər).

Məsələn:

```
const  Max = 150;
const float  Pİ= 3.14159 ;
```

Yə'ni,Max dəyişəni proqramda 150 qiyməti alır və dəyişə bilməz.

Dəyişənin tipi volatile olduqda onun qiyməti nəinki proqramçı tərəfindən həm də sistem tərəfindən dəyişdirilə bilər. Dəyişənin tipinin void olması onun tə'yin olunmadığını və onun gələcəkdə tə'yin olunacağını göstərir. Funksiyanın tipi void-lə tə'yin olunarsa nəticənin funksiyasının tipinə bağlı olmadığını göstərir və funksiya return ilə verilmir (növbəti mövzularda araşdıracağıq). C++ dilində dəyişənin bilavasitə yerləşdiyi ünvana müraciət etmək üçün xüsusi pointer (göstərici) tip dəyişənlərdən istifadə edilir. Göstərici tip dəyişənlərdə hər hansı bir dəyişənin ünvanı yerləşir. Yə'ni göstərici tip dəyişəndə verilmiş hər hansı bir qiymət ünvan kimi başa düşülür. Ümumi yaşılışı:

<dəyişənin tipi> * [modifikator] < dəy. adı>

kimidir.

Məsələn:

```
1) file  * Mas ;      (fayl tipli göstərici)
char   * ALFA ;      (simvol tipli göstərici)
char   * MAS [20] ;    (massiv tipli göstərici)
int (*GAM ) ( ) ; (funksiya tipli göstərici)
2) int x ;
int  *y = &x ;
 x = 5;
*y = 20;
cout <<" \ n x =  "<< x;
```

Proqram fraqmenti yerinə yetirildikdə ekranda x=20 əks olunur.

Əgər göstərici tip dəyişənin tipi qeyri müəyyən olarsa (yə'ni void) onda bu dəyişəndən hər dəfə istifadə etdikdə onun tipi aşkar şəkildə əməliyyatlarda göstərilməlidir.

Məsələn:

```
float n ;
void  *un ;
un =  &n ;
( float *) un ++ ;
```

Burada ixtiyari tipdən olan (void) un dəyişəni göstərici kimi e'lan olunmuşdur. Ona görə də bu göstəricinin tipi aşkar olunmamış əməliyyat aparmaq olmaz. (float*)- dan istifadə edərək un dəyişənin qiymətini bir vahid artırırıq.

Göstərici tipdə modifikator kimi const, near, far, huge açar sözlərindən istifadə edə bilərik. Bu atributlar göstərici dəyişənin neçə baytda yerləşdiyini tə'yin edir.

const– göstəricini proqramda dəyişməyəcəyini göstərir.

near– atributu iki baytlıq göstərici dəyişəni tə'yin edir. Başqa sözlə nisbi ünvanı göstərir. Bu atributla yalnız bir seqment daxilindəki ünvana müraciət etmək olar.

far – atributu 4 baytlıq ünvan dəyişənini tə'yin edir. Belə dəyişən 1mb yaddaş ünvanını tə'yin edə bilər. Bu atributla tə'yin olunan ünvan aşağıdakı kimi hesablanır.

Seqment ünvanı x 16 +sürüşdürmə

Baxmayaraq ki, far atributu 4 baytlıq ünvan dəyişənini təyin edir, onun qiyməti yalnız seqment daxilində dəyişə bilər. Yə'ni digər seqmentə müraciət edə bilməz.

huge– atributu 4 baytlıq dəyişəni tə'yin edir. Bu atributun far-dan əsas fərqi ondadır ki, bu atributla ixtiyari seqmentə müraciət edə bilərik .

məsələlərə baxacayıq.

Ifadələr

C++ dilində də başqa dillərdə olduğu kimi ifadələrdə sabitlərdən, dəyişənlərdən, funksiyalar və onlar arasındakı əməliyyatlardan istifadə edilir. C++ dilində əməliyyatlar bir operandlı (unar), iki operandlı (binar) və üç

operandlı (ternar) əməliyyatlara bölünürlər. Unar əməliyyatlar aşağidakı cədvəldə verilmişdir.

İşarə	Əməliyyatın adı	Əməliyyat qrupu
! ~ -	Məntiqi inkar Əks kodun alınması, mərtəbə-mərtəbə inkar Hesabi inkar	İnkar və tamamlama
* &	Ünvandakı məzmun Ünvanın təyini	Unvandakı məzmun və unvan
+	unar toplama	unar toplama
sizeof	Operantın ölçüsünü təyini	Ölçü

Binar və ternar əməliyyatlar aşağıdakı kimidir:

İşarə	Əməliyyat	Əməliyyatın qrupu
+ -	Toplama Çıxma	Additiv
* / %	Vurma Bölmə Qalıq hissənini tə'yini	Multiplikativ
<< >>	Sola sürüşdürmə Sağa sürüşdürmə	Sürüşdürmə
< > ==	Kicik Böyük Bərabər	Münasibət

<= >= !=	Kiçik bərabər Böyük bərabər Bərabər deyil	
& : ^	Mərtəbə-mərtəbə məntiqi və Mərtəbə-mərtəbə məntiqi və ya Mərtəbə-mərtəbə məntiqi və yanın inkarı	mərtəbə-mərtəbə
&& ::	Məntiqi və Məntiqi və ya	Məntiqi
,	Vegül (ardıcıl hesablamalar)	ardıcıl hesablamalar

Aşağıda mənimsətmə əməliyyatlarının cədvəli verilmişdir. Mənımsətmə əməliyyatlarında sağ tərəfdəki operandın qiyməti sol tərəfdə dayanan operand ilə təyin edilən dəyişənə uyğun yaddaş sahəsinə yazılmalıdır. Ona görədə mənimsətmə əməliyyatlarında sol tərəfdə dayanan operand yaddaş sahəsinə müraciət edən ifadə olmalıdır.

İşarə	Əməliyyatın adı
++	Artırma (unar inkriment)
--	Azaltma (unar dekrement)
=	Bərabərlik
*=	Vurulma mənimsətmə ilə
/=	Bölmə mənimsətmə ilə
%=	Qalıq hissənin seçmə mənimsətmə ilə
+=	Toplama mənimsətmə ilə
-=	Çıxma mənimsətmə ilə

<<=	Sola sürüşdürmə mənimsətmə ilə
>>=	Sağa sürüşdürmə mənimsətmə ilə
&=	Məntiqi və mənimsətmə ilə
:=	Məntiqi və ya mənimsətmə ilə
^=	Məntiqi və yanin inkarı mənimsətmə ilə

Proqramın strukturu

C++ dilində yazılan poqram ixtiyari sayda operatorlardan, funksiyalardan və bloklardan təşkil edilə bilər. Lakin proqramın daxilində yalnız bir dənə main tip funksiya ola bilər. main() funksiyası proqramın ixtiyari yerində yazıla bilər. Hətta proqram ayrı–ayrı ilkin modullar şəklində yazıla bilər. Sonra isə xüsusi redaktorun köməyi ilə birləşdirilə bilər. Proqrama daxil olan baş funksiyadan başqa digər funksiyalar ayrı-ayrılıqda translyasiya edilib, ayrıca obyekt faylı kimi saxlanıla bilər. Lazım gəldikdə xüsusi əmrin vasitəsilə proqrama birləşdirilər. Şərti olaraq C++ dilindəki proqramın ümumi strukturasını aşağıdakı kimi göstərmək olar.

```
[qlobal dəyişənlərin e'lanı]
[funksiyaların prototipllərinin e'lanı]
main ( )
  {
   lokal dəyişənlərin e'lanı
   operator;
   operator;
    …
  }
```

```
<funksiya>
[lokal dəyişənlərin e'lanı]
{
…
}
…
```

Ümumi strukturadan göründüyü kimi baş funksiya və ya ixtiyari funksiya bir və ya bir neçə blokdan ibarət ola bilər. Blokun başlanğıcı {mötərizə, sonu isə} mötərizə ilə qurtarır. Başqa dillərdə olduğu kimi C++ dilində də bir blokun daxilində bir neçə blok ola bilər. Bu bloklar bir-birindən daxili və xarici bloklar kimi fərqlənirlər. Hər bir blok daxilində müəyyən dəyişənlər e'lan oluna bilər. Daxili blokda e'lan edilən dəyişən xarici bloka nəzərən lokal dəyişəndir. Yə'ni o xarici blokda öz qiymətini saxlamır. Qlobal dəyişənlər xarici bloklarda və ya main funksiyasından əvvəl e'lan olunur. Qlobal dəyişənlər təsir etdiyi bloklarda öz qiymətlərini saxlayırlar. Lokal dəyişənlərin qiymətlərini yadda saxlanılması üçün, yə'ni yenidən həmin bloka qayıtdıqda, qiymətini saxlaması üçün onların yaddaş sinifi static kimi tə'yin edilməlidir.

Ümumi strukturaya aid misala keçməzdən öncə sadə çıxış operatoru olan printf-i nəzərdən keçirək.

printf funksiyası standart proqramxanaya daxil olan funksiyadır. Bu funksiya stdio.h proqramxanasında yerləşir. Bu proqramxanadakı funksiyaların proqrama qoşulması üçün include operatorundan istifadə edilir (gələcək mövzularda tam araşdıralacaq). printf funksiyanın köməyi ilə sabitlərin və ya dəyişənlərin qiymətlərini ekrana vermək olar. Bu funksiyanın ümumi yazılışı aşağıdakı kimidir:

```
printf ("SP" , DS);
```

Burada SP - spesifikator olub sabit və dəyişənlərin formatlarını tə'yin edir;

DS - ekrana veriləcək dəyişənlər siyahısıdır.

Dəyişənlərin qiymətlərini ekrana vermək üçün müxtəlif formatlardan istifadə edilir. Hər formatın önündə % işarəsi yazılır.

Spesifikatorlarda xüsusi idaraedici işarələrdən istifadə edilir (eskeyp ardıcıllıqdan). Spesifikatorun daxilində işarə tip sabitlərdən də istifadə etmək olar.

printf funksiyasında əsasən aşağıdakı spesifikatorlardan istifadə edilir:

Format	Obyektin tipi
%c	Char
%s	Sətir
%d	int (onluq)
%o	int (səkkizlik)
%u	Unsigned int
%x	int (onaltılıq)
%ld	long int (uzun onlu)
%lo	long int (uzun səkkizlik)
%lu	Unsigned long
%lx	long (uzun onaltılıq)
%e	float/double (sürüşkən nöqtəli)
%g	float/double (qiymətdən asılı olaraq f və ya e formatında)
%lf	long float (sabit nöqtəli)
%le	long float (sürüşkən nöqtəli)
%lg	long float (f və ya e formatında)
%n.mf	Sabit nöqtəli həqiqi verilənlər. Burada n ədədin tam hissəsindəki rəqəmlərin sayı, m isə mantissadakı rəqəmlərin sayıdır.

İndi isə sadə bir proqramı nəzərdən keçirək.

```
# include  <stdio.h>
   int       i=1 ;
main ( )
{    /*   1-ci blok */
printf (“%d\n”, i) ;
{  int  i=2 ;
      /* 2-ci blok */
printf (“% d\n”, i) ;
{ int  i=3 ;
/* 3-cu blok */
printf (“% d\n”, i) ;
}
printf (“% d\n”, i) ;
  }
}
```

Ekranda aşağıdakı verilənlər çap olunacaq.

```
1
2
3
  2
```

Klaviaturadan formatlı daxiletmə. Klaviaturadan formatlı daxiletmədə scanf() funksiyasıdan istifadə olunur. Bu funksiyanın çağırılma operatorunun strukturu aşağıdakı kimidir:

scanf(format_sətri, arqumentlərin_siyahısı)

scanf() daxiletmə funksiyası klaviaturadan daxil edilən simvolların oxunması və onların tipə uyğun daxili təsvirə çevrilməsini həyata keçirir. scanf() funksiyasında format sətri və arqumentlərin siyahısı hökmən olmalıdır.

scanf() funksiyasının sətrində hər bir daxil edilən dəyişən spesifikasiyaya uyğun olmalıdır. Burada hər bir dəyişənin adının qarşısında & simvolu qoymaq lazımdır. Bu simvol “ünvanı götürmək” deməkdir.

Misal :

```
#include<stdio.h>
main()
{
 int e,u;
printf(“Düzbucaqlının enini daxil edin”);
scanf(“%d”,&e);
printf(“Düzbucaqlının uzunluğunu daxil edin”);
scanf(“%d”,&u);
printf(“\n\n En=%d, Uzunluq=%d,Sahe=%d\n”,e,u,e*u);
}
```

Misalda “%d”–format sətri, &e –isə arqumentin siyahısıdır. Bu operator e dəyişəninə ədədi qiymətin daxiledilməsinə imkan verir. Daxiletmə scanf() funksiyasında h, I, L modifikatorlardan istifadə etmək mümkündür. Bunlar daxiletmədə dəyişdirilmiş tiplərdən istifadəyə imkan verir. Tiplər aşağıdakılardır:

hd- short int tipli qiymətləri daxiletmək üçün;

ld- long int tipli qiymətləri daxiletmək üçün;

lf, le- double tipli qiymətləri sabit və sürüşkən nöqtəli formada daxiletmək üçün;

Lf, Le-long double tipli qiymətləri sabit və sürüşkən nöqtəli formada daxiletmək üçün.

Yuxarıdakı proqramdakı iki dəyişənin qiymətlərini bir operatorla daxil etmək mümkündür:

```
scanf(“%d%d”,&e,&u);
```

Qeyd edək ki, müxtəlif qiymətlər arasında istənilən sayda boşluq, həmçinin tabulyasiya və sətrin simvollarından da istifadə etmək olar.

Axınlı daxiletmə-xaricetmə. C++ dilində daxiletmə-xaricetmənin spesifik vasitələri mövcuddur. Bu <iostream.h> faylının köməyi ilə proqrama qoşulan *siniflər kitabxanasıdır*. Bu kitabxanada standart simvollar axınının obyekti kimi aşağıdakı adlar təyin edilib:

cin-klaviaturadan standart axınlı daxiletmə;

cout- ekrana standart axınlı xaricetmə.

Verilənlərin daxiledilməsi axından cin-i çıxarmaq və dəyişənlərə uyğun qiymətləri mənsubetmə kimi interpretasiya olunur. C++ dilində *standart axından* çıxarmaq əməliyyatı >> işarəsi ilə təyin olunub. Məsələn: z dəyişənin qiymətinin daxiledilməsi aşağıdakı kimidir:

```
cin>> z;
```

Verilənlərin xaricedilməsi standart axından cout-la birlikdə xaricedilməsi qiymətləri yerləşdirmək kimi interpretasiya olunur. *Standart axında* yerləşdirmək əməliyyatı << işarəsi ilə təyin olunub.

Məsələn:

```
cout<<x*z;
cout<<"\nCavab="<<Y';
cout<<"e="<<e<<"u="<<u<<endl;
```

Misallardan göründüyü kimi axında printf() funksiyasında olduğu kimi idarəedici simvollardan istifadə etmək olar. Buradakı endl manipulyatordur və kursorun yeni sətrə keçirilməsini(\n idarəedici simvolu kimi) təyin edir. cin və cout haqqında sonra gəniş məlumat veriləcək.

Tə'yin edilən funksiya hər hansı bir işi yerinə yetirməsi üçün, o dəyişənlərdən istifadə etməlidir. Əvvəkki mövzularda dediyimiz kimi C++ dilində dəyişənlərin tipi onlardan istifadə edilənə qədər tə'yin edilməlidirlər. Tipin tə'yini dəyişən üçün uyğun yaddaş sahəsinin ayrılması deməkdir. Proqram bloku anlayışı ilə əlaqəli dəyişən üçün tə'yin edilən yaddaş sinfi onun görünmə oblastını və yaşama müddətini tə'yin edir. C++-da blok dedikdə {,} mötərizələri arasına alınmış e'lanlar, tə'yinatlar, və operatorlar ardıcıllığı başa üşülür. C++-da iki cür blok var: mürəkəb operator və funksiya. Bloklar bir-birindən daxili və xarici bloklar kimi fərqlənirlər.

Dəyişənin yaşama müddəti dedikdə proqramın icrası zamanı dəyişənin var olduğu intervala deyilir. Dəyişənin yaşama müddəti qlobal və ya lokal ola bilər. Dəyişən qlobal işə onun yaşama müddəti proqramın tam icra müddətidir.

Lokal dəyişənlərin yaşama müddəti isə tə'yin olunduğu blokların icra müddətidir.

Funksiyalar proqramda qlobal yaşama müddətinə malikdirlər, yə'ni proqramın tam icra müddətinə.

Obyektin istifadə olunduğu proqram hissəsinə onun görünmə oblastı deyilir. Əgər obyektin blokda və ya proqramda adı və tipi tə'yin edilibsə onda o, blokda və ya funksiyada görünən hesab edilir. Əgər obyekt blokun daxilində e'lan edilibsə o, ancaq blokda və blokun daxili bloklarında görünəcək. Əgər obyekt yuxarı səviyyədə e'lan edilibsə o həmin nöqtədən priqramın sonuna qədər görünəcək.

Ə ə ş ə ə ş
ü ə ə Ə ə ş ə
ə' ə ş ü ə ə ə ş
ə ə' 1 ' 1 ə ə ə ə 1
ə ə ə ə şə 1 1 1 1

Dəyişənlərin daxili və xarici səviyyədə elanı

ə ə ə ə ə ə ' 1

Dəyişənin daxili səviyyədə e'lanında dörd yaddaş sinfinin dördündən də istifadə edə bilərik. Əgər daxili səviyyədə dəyişənin e'lanı zamanı yaddaş sinif spesifikatoru göstərilməyibsə yaddaş sinfi auto kimi tə'yin edilir. auto yaddaş sinfi ilə tə'yin edilən dəyişən lokal yaşama müddəti ilə tə'yin olunur. Yuxarda dediyimiz kimi belə yaddaş sinfinə malik dəyişənlər üçün bloka daxil olma zamanı yaddaş sahəsi ayrılır, çıxarkən yaddaş azad edilir. İkinci dəfə bloka daxil olarkən dəyişən üçün başqa yaddaş sahəsi də ayrıla bilər.

register yaddaş sinfli dəyişən üçün kompilyator imkan daxilində ümumi tə'yinatlı registerlərdən birini yaddaş kimi ayırır. Əgər belə ayrılma mümkün isə əməliyyatlar sür'ətlə aparılır. register yaddaş sinfi ilə tə'yin edilən dəyişənlərin görünmə oblastı auto yaddaş sinfində olduğu kimidir. Kompilyator register yaddaş sinifli dəyişənə rast gələrkən bu zaman boş register yox isə, onda dəyişən üçün auto yaddaş sinifi tə'yin edillir. Əgər

register yaddaş sinfi mümkün isə, o ancaq int və ya int tip uzunluqlu göstərici tip dəyişən üçün ayrılır.

Əgər daxili səviyyədə dəyişəni static yaddaş sinfi ilə tə'yin etmişiksə, onda blokdan çıxdıqda və yenidən qayıtdıqda dəyişənin qiyməti dəyişməz qalır.

auto yaddaş sinifli dəyişənlər üçün yer stek yaddaşda ayrılır (ona görə də blokdan çıxdıqda bu dəyişənlərin qiyməti itir). static yaddaş sinifli dəyişənlər üçün isə yer verilənlər seqmentində ayrılır və buna görə də blokdan çıxdıqda qiymətləri saxlanılır.

extern yaddaş sinfi ilə e'lan edilən dəyişənə həmin ad ilə yuxarı səviyyədə tə'yin edilən istənilən proqramdan müraciət etmək olar. Dəyişəni daxili səviyyədə extern kimi e'lan etmək ona görədir ki, xarici səviyyədəki tə'yinatı daxildə görünən etsin. Xaricdə extern kimi e'lan edilməyən və yalnız daxildə extern kimi tə'yin edilən dəyişən ancaq tə'yin olunduğu blokda görünən olacaq.

ə şə ə ə ə ' 1

ə şə ə ə ə ə 1 _____ ə _____ ş ə ə '

ə ə ə ' ə ə ə ə ə ə ə ə şə üçü

ş ə ə' ə ə

ə şə ə ə ə ə ' 1 ə şə ə' ə ə

ə ə' ş ə şə ə 1 1 ə ə ə ə' ş ğı 1

ü ə ə

ə şə 1 ə ə ə ____ ş ə ə' ə ə ə

ə şə ş ə ə ə ə ə ə

Ə ə ş ə ə ö ə ə 1 1 ö ü ü ü ə'

ə

ə ə

Ə ə ə şə 1 ə ə ə ə 1 ə' ş

ə ə ə

' ə " " ə şə ə' ğ ö ə

Ə ə ə şə ə ə ə ə’ ə ə’ ğ 1
1 ə ə ə ə ö ü ə Ə ə ü ə ə’
ə ə’ ğ ə ə ə ə ə ş ö ü ə

ə şə ə 1 ç ə 1 ə ə ə ə ə ’ ə
ə ə şə ə ə’ ə ö ə ə
_____ ş 1 ö 1 ə ö ü ə
_____ ş ə ’ ş ə ə ə ə şə ə ü ə ə ə

_____ ş ə ’ ə ə şə ə ə ə çü ü
ə ə ə ə’ ş ə şə ə ü ə ə

Sadə hesabi çevrilmələr Multiplikativ əməliyyatlar. İnkrement və Dekrement.

İfadələr icra olunarkən dəyişənlərin tiplərinin avtomatik çevrilməsi baş verir. Yerinə yetirilən çevrilmələr əməliyyatın xüsusiyyətindən asılıdır və oprerandın tipindən asılıdır. Belə çevrilmələrə hesabi çevrilmələr ona görə deyirlər ki, onlar adətən hesab əməliyyatları zamanı baş verirlər.

Sadə hesabi çevrilmələrdə aşağıdakı qaydalara rəayət olunur.

1. float tipli operandlar double tipə çevrilirlər.
2. Əgər operandlardan biri long double isə onda digəri də long double tipə çevriləcək.
3. Əgər operandlardan biri double isə onda digəri də double tipə çevrilər.
4. İxtiyari char və ya short tipdən operand int tipə çevrilirlər.
5. İxtiyari unsigned char və ya unsigned short tipdən operand unsigned int tipə çevrilirlər.
6. Əgər operandın biri unsigned long isə digəri də unsigned long tipə çevrilir.
7. operandlardan biri long isə digəri də long tipə çevrilir.

8. Operandlardan biri unsigned int isə digəri də unsigned int tipə çevrilir.

Beləliklə hesablama zamanı operandlar ən uzun tipli operanda çevrilirlər.

Aşağıdakı misalı nəzərdən keçirək:

```
double f;
 unsigned char c;
unsingned long m;
int i;
```

olarsa.

f*(i+c/m) ifadəsi hesablanarkən aşağıdakı çerilmələr baş verəcək.

1. c operandı əvvəlcə 5-ci qaydaya görə unsingned int çevriləcək.

Sonra isə 6 -cı qaydaya görə unsigned long- a çevriləcək.

1. 6-cı qaydaya görə i operandı unsigned long-a çevrilir və beləliklə bütün mötərizədəki ifadə unsigned long tipində olur.
2. 3-cü qaydaya görə ifadə double-yə çevriləcək.

Əməliyyatların nəticələri haqqında qısa məlumat verək.

1. - -operandı əks işarəli operand kimi verir. Operand tam və ya sürüşkən vergüllü ola bilər.

Məsələn:

```
int i=51;
 i= -i;
 i= -4;
```

2. '!'- operandın məntiqi inkarını qaytarır. Yə'ni operand "doğru" isə 0, "yalan" isə 1 qaytarır. Nəticə int tipdən olmalıdır. Operand isə tam həqiqi və ya göstərici tipdən ola bilər.

Məsələn:

```
int x=0;
if (!x);          yoxlamasının nəticəsi doğru olacaq.
```

3. ~ - ikilik tamamlama əməliyyatı operandın ikilik əks kodunu qaytarır.

Məsələn:

```
main ( ) {
char c = '9';
```

```
unsigned char cc;
cc = ~ c;
cout << “c (simvol)= “<< c << ” \ t c ( onaltılıq )= ” << c <<endl;
 cout << “c (simvol)= “ << cc << ” \ t ~ c ( onaltılıq )= ” <<cc << endl;
}
```

Nəticədə ekranda

c(simvol) =9 c(onaltılıq) =39

~c(simvol) = Ə ~c(onaltılıq) =c6 alarıq.

4. * - əməliyyatında operand göstərici olmalıdır. * əməliyyatı ünvanlaşdırılan kəmiyyətə göstərici vasitəsilə bilavasitə müraciəti təşkil edir. Nəticə göstərilən ünvandakı kəmiyyət olur. Nəticənin tipi göstərilən ünvandakı kəmiyyətin tipindən olur.

Əgər göstərici mümkün olmayan ünvana müraciət edərsə nəticə tə’yin olunmaz. Belə vəziyyətlər aşağıdakılardan biri ola bilər.

1. göstərici sıfıra bərabərdir.
2. Aktiv olmayan lokal obyektin ünvanına müraciət olunanda.
3. Göstərici obyektin uyğun olmayan düzləndirilməmiş ünvanına müraciət etdikdə.
4. Göstərici proqramda istifadə olunmayan ünvanı tə’yin etdikdə.

& - əməliyyatı operandın ünvanını tə’yin edir. Operand kimi adı olan ifadə, funksiyanın adı və massivin adı ola bilər(baxmayaraq ki, axırncılar özləri ünvanı göstərir). & əməliyyatının nəticəsi operandın göstəricisi olacaq. & əməliyyatın bitlər sahəsindən ibarət strukturaya və register yaddaş sinifli obyektlərə tətbiq edilə bilməz. Məsələn:

```
int x, y=10, *adres ;
adres = &x ;
*adres = y ;
```

sizeof əməliyyatı ilə identifikator və tip üçün ayrılan yaddaş sahəsinin ölçüsünü tə’yin edir. Yazılşı:

sizeof <ifadə> kimidir.

Burada <ifadə> (,) mötərizələri arasına alınmış identifikator və ya tipin adı ola bilər. Tip adı kimi void götürmək olmaz. <ifadə> kimi bitlər sahəsi və ya funksiyanın adını götürə bilmərik.

Strukturarın və ya birləşmə tipin ölçüsünü bu əməliyyat ilə tə'yin edərkən bu ölçüyə onlara daxil olan elementlərin düzləndirilmiş sahələri daxil edilir. Ona görə də bu ölçü ilə strukturanın ayrı-ayrı elementlərinin ölçülərinin toplanması üst-üstə düşməyə bilər.

Məsələn:

```
main() {
struct { char a; int b; double f;
} str;
cout << " str dəy.ölçüsü = "<< sizeof (str);
}
```

nəticə str dəy.ölçüsü=12 olur. (sadə hesabat isə 11 vərər)

Multiplikativ əməliyyatlar

*, /, % əməliyyatları multiplikativ əməliyyatlar adlanır. % əməliyyatının operandları tam ədədlər olmalıdır. * və / əməliyyatları isə həm tam həm də sürüşkən nöqtəli operandlar üzərində aparıla bilər. Bu əməliyyatlarda operandlar adi hesabi çevrilmələr ilə aparılır. Nəticənin tipi operandların çevrilməsindən alınan tipdir.

Qeyd: bu çevrilmələrdə daşma və işarənin itməsi vəziyyətləri nəzərə alınmır. Bu da informasiya itkisinə səbəb ola bilər.

* əməliyyatına aid misalı nəzərdən keçirək

```
 int    a= 5;
float       y= 0.2;
double      x, z;
   x=y*a;   z= y*(x – a);
```

1 – ci əməliyyatda y və a dəyişənləri double tipə çeviriləcəklər sonra isə * əməliyyatı aparılacaq.

2 - sində əvvəlcə a double tipə çevrilək (x-a) əməliyyatı aparılacaq. Sonra isə y double tipə çevrilərək * əməliyyatı aparılacaq.

/ - bölmə əməliyyatı zamanı əgər 1 – ci operand 2 – ciyə tam bölünməzsə nəticə aşağıdakı qayda ilə tə'yin edilir.

1. Əgər operandlar unsigned tipli müsbət verilənlər isə kəsr hissə atılır.
2. Sıfra bölmə zamanı bu haqda xəbər verilir.

% -qalığın tapılması zamanı qalığın işarəsi 1 – ci operandın işarəsinə uyğun götürülür.

Məsələn:

```
main()
{
int    a= 5;
int    y= -2;
cout <<“ a/y= ”<<a/y;
cout <<“\t a%y= ”<<a%y;
}
```

Proqramının icrasından sonra ekranda

a/y= -2 a%y=1 əks olunur.

Additiv əməliyyatlar tam və sürüşkən nöqtələri verilənlər üzərində aparıla bilər. Bə'zi hallarda bu əməliyyatlarda operand kimi göstərici tipi də götürə bilərik.

Bu əməliyyatlarda da adı hesabi çevrilmələr aparılır. Additiv əməliyyatlarda çevrilmələr zamanı baş verə biləcək daşma və ya qiymətin itməsi vəziyyətləri nəzərə alınmır. Əgər çevrilmə zamanı tipə uyğun təsvir mümkün deyilsə informasiya itir.

Məsələn:

```
main ( )
{          int a = 30000 ;
           int b = 30000 ;
           int c ;
            c = a + b ;
```

```
cout << " a =" <<a <<" \ t  b = " <<b <<" \ t a + b =  " <<c;
        }
```

Nəticədə

A = 30000 b = 30000 a +b = -5536

düzgün olmayan nəticə alırıq.

Struktur tipli dəyişənlər

Müxtəlif tipli dəyişənləri eyni ad altında birləşdirmək üçün struct tipli dəyişənlərdən istifadə olunur. Aydındır ki, bu tip verilənlər eyni obyekti xarakterizə etməlidirlər. struct tip verilənlər {, } sistem mörtərizələri içərisində yazılır və ümumi yazılışı aşağıdakı kimidir.

```
struct [TA]
{
  <ST1>    <SA1> ;
  <ST2>    <SA2> ;
          - - -
  } [DA] ;
```

Burada TA – struct tipin adı;

ST1, ST2, … - strukturaya daxil olan sahələrin tipləri;

SA 1 , SA 2 , … - strukturanın sahələrinin adları;

DA – struct tip dəyişənin adını tə'yin edir.

Məsələn:

```
1).  struct A  {
          İnt , x ;
        unsigned  y ; };
2).  struct  BB {
          char N ;
    int L ;
    float  k ;  };
```

Aşağıdakı verilənlərdən ibarət mürəkkəb strukturanı nəzərdən keçirək.

1. Familiyası (Soy_adı 20b).
2. Adı (ADI 15b).
3. Atasının adı (ATA_ADI 15b).
4. Anasının adı (ANA_AD 15b).
5. Doğulduğu yer (D_YERI 20b).
6. Doğum tarixi (DOĞ_TAR 6b).
7. Qan qrupu (QAN 8b).
8. Dini (DİN 15b).

Bu strukturanı bir-birinin daxilinə qoyulan struktura kimi yazaq.

```
struct  D_TAR {int Gun, Ay, İL ; }; və
        struct   Shahs _QEYDİ {
            char SOY_adı [20];
            char ADI [15], ATA_AD [15], ANA_AD [15];
            char    D_YERI [20];
            struct   D_TAR   DOG_TAR;
            char    QAN[8], DIN [15];
};
```

Shahs strukturunun daxilində strukturanın aşağıdakı yazılış formasından istifadə edilmişdir.

srtuct <st> <da1> [,<da2>]…

burada, st əvvəlcədən təyin edilmiş strukturanin tipini,

da1, da2… həmin tip ilə təyin edilən struktur tip dəyişənlərin adını göstərir.

Məsələn:

```
struct c c1,c2;
```

yazılarsa, bu o deməkdir ki, c1 və c2 dəyişənləri c tip struktura ilə tə'yın olunmuşdur. Yə'ni c strukturasının hər hansı k-cı elementinə c.k kimi müraciət ediriksə, onda c1.k yazılışı da səhv deyil.

Bu isə o deməkdir ki, eyni strukturaya malik dəyişənləri bir dəfə elan etmək kifayətdir.

Qeyd: <u>funksiyada struktur tip verilənlər faylın əvvəlində təsvir edilir.</u>

Strukturanın adı ilə dəyişənlərin adı eyni ola bilər. Bəzi məsələlərdə ikilik mərtəbədən (bitlərdən) istifadə edilir. Belə halda strukturanın sahələri aşağıdakı kimidir:

<st > da:di;

burada – st səhənin tipini;

-da sahəni göstərən dəyişənin adını;

di-sabit tam ifadəni göstərir.

Sahənin tipi ,ancaq signed və ya unsigned kimi təyin edilə bilər. Dəyişənlərə ad verilməsində məqsəd, ancaq sahələrin düzləndirilməsi üçündür. Məsələn: tutaq ki, İki baytlıq yaddaş sahəsində kompüterin müəyyən qurğuları haqqında məlumat vardır.Bu məlumat yaddaş sahəsində aşağıdakı kimi yerləşmişdir.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y	Y		O	P	P	P		S	S	E	E			C	F

Burada

YY-sahəsində kompüterə qoşulan printerlərin sayı təyin edilir(nam_print);

O- qoşulan oyun adapteri haqqıda məlumatı saxlayır(game_adap);

PPP- R232 portunun miqdarı (ports);

SS- sistemlə baglı sürücü miqdarı(num_flop);

EE- ekranın modulunu (video_mode);

C- soprosessor (S8087);

F- disket (flop_boot).

Bu verilənləri aşağıdakı struktura ilə təyin edə bilərik

```
struct  eq {
 int     floop_boot    : 1 ;
 int     S8087          : 1 ;
 int                    : 2 ;
 int     video_mode  : 2 ;
 int     num_flop       : 2 ;
```

```
                int                     : 1 ;
                int     ports           : 3 ;
                int     game_adap       : 1 ;
                int                      : 1 ;
                int     num_print       : 2 ;
                 }  eq;
```

Bu sturukturadan proqramda aşağıdakı kimi istifadə edək.

```
#include <iostream.h>
 #include <bios.h>
 int biosequip(void);
 void main()
 {
struct eq {
 unsigned flop_boot       : 1;
 unsigned s8087            : 1;
 unsigned                  : 2;
 unsigned video_mode   : 2;
 unsigned num_flop        : 2;
 unsigned                  : 1;
 unsigned ports            : 3;
 unsigned  game_adap    : 1;
 unsigned                   : 1;
 unsigned   num_print     : 2;
 }eq;
  int *i;
 i=(int*) &eq;
 *i= biosequip();
 cout <<"\n disk qurqularinin.sayi="<<eq.num_flop;
 cout <<"\n portlarin sayi="<<eq.ports;
 if (eq.s8087) cout <<"\n soprosessor var ";
```

```
else cout <<"\n soprosessor yoxdur";
}
```

Hər bir dəyişəni bir tam ədədə uyğun gələn nömrələnmiş çoxluğa sadalanan tip (enum) deyilir. Sadalanan tip dəyişənlərin ümumi yazılışı aşagıdakı kimidir;

enum [TA] { DS }DA ;

Digər yazılışı

enum < TA> DS ;

kimidir.

Burada TA –tipin adını;

DS – bir–birindən vergül ilə ayrılan dəyişənləri göstərir.

Birinci yazılış formasında dəyişən dəy=<sabit ifadə> kimi də tə'yin edilə bilər.

Dəyişənlər int tipdən olmalıdır və 2 baytlıq yaddaş sahəsi tuturlar. Dəyişənlərin adları unikal olmalıdırlar. Dəyişənlərə aşkar surətdə qiymətlər mənimsədilməmişsə onda 1- ci elementə susmaya görə sıfır mənimsədilir. Növbəti elementlərin qiymətləri bir vahid artır. <dəy adı>= <sabit ifadə> yazılışından sonrakı element sabit ifadənin qiyməti bir vahid artırılır (əgər o, yeni sabit ifadə ilə qiymətləndirilmirsə).

Dəyişənlərin qiymətləndirilməsi zamanı aşağıdakı qaydalar var:

1. Sadalanan dəyişənlər eyni qiymətlər ala bilər.

2. Dəyişənlərin adları başqa sadalanan tipin adlarından və oblastın bütün adlardan fərqli olmalıdır.

Məsələn:

```
enum d {
        s 1,
        s 2,
        s 3 = 15,
```

s 4,

s 5 };

Burada s1 =0; s2 = 1; s3 = 15; s4 = 16; s5 = 17 qiymətlərini alırlar.

İş ə ə ə

C++ dilində işarələr sətrini 2 şəkildə vermək olar: Massiv və ya göstərici (pointer). Aşağıdakı strukturanı nəzərdən keçirək.

1. Tələbənin familyası
2. Adı
3. Atasının adı

Strukturanın verilənlərinə uyğun olraq aşağıdakı adları verək və yaddaş sahələri ayıraq.

1. Fam (20b)
2. AD (15b)
3. A_ ad (15b)

Onda bu dəyişənləri massiv kimi:

char FAM [20];

char AD[15];

char A_AD[15]; e'lanları ilə tə'yin edə bilərik

Göstərici kimi isə

char * FAM;

char * AD;

char * A_AD; tə'yin edə bilərik.

Qeyd: İşarələr sətrinin sonu \0 ilə bitməlidir. Belə ayırma ada bağlı deyil və massivdə aşağıdakı kimi göstərilir.

Məsələn: char AD [10] ;

e'lan etsək və bu sahəyə "Ulkar" və ya "Camaladdin" adlarını yazaq. Onda virilənlər yaddaşda

U	L	k	A	r	\0	\0	\0	\0	\0

C	A	m	a	L	a	d	d	i	n	\0

kimi olar.

pointer-lə isə yə'ni, char * AD ; ilə

U	l	k	A	r	\0

C	A	m	A	L	a	d	d	i	n	\0

olar.

İşarələr sətrindən fərqli olaraq işarə tip dəyişənin sonunda \0 qoyulmur. Yə'ni əgər "x" işarələr sətri isə bu yaddaşda □

	\0

olar. Əksinə ' x ' işarə tip isə yaddaşda kimi olar.

Münasibət əməliyyatları, mərtəbə məntiqi əməliyyat, məntiqi əməliyyatlar

ü ə ə ə 1 ə ü ə

Ə ə ü ə ğ ə ə ə ğ ə ə ə ə 1 1

ö ü ü ü ə ə ö ə ə ə ə ə

ə 1 ü ə ə ə ə ə ə ə

ü ə ə ç ə ə 1 ə 1 ş ğ1 1

ə

ə ə ö ə ə ə ə ə

ə ə ü ə ə ə ə ə ö ə ş

1 ü ş 1ğ1 1 ə' ş ü ə ə ə

ş ü 1 1 ə ə ə ə ö ə' ə

ə ü 1 ğ ö ə ə ə ə

ə ə ə ü 1 1 ü ə ə’ 1 1

ə ü 1 1 ü ə ə

ö ə ə ə ə 1 ə ü ə ə ə

ö ə ə ə ö ə 1 ə ş 1 1

ö ə

ü əə ə 1 ş ğı 1 ə ə ə ç ə

…

” ”

ə ə 1 1 öü ə ə ə ə ç 1 əş

ə ə ə– ə ə ə ə ə ə

ə ə ə– ə ə ə ə ə ə 1 1

ə ə ə ə ç əə 1 ə

ə ə 1 ç ə ə 1

ə ə 1 – 1 ə – 1 ğ ə

ü ə Ə ə ə ə ə ə ğ ə

ə ə 11

ə ə 1 – 1 ə – 1 ğ ə

ü ə ü ə ə ə ə ə ə ə ə

11 öüüü

ə ə 1 – 1 ə – 1 ğ ə

ü ə Ə ə ü ə ə ə ə ə 11 ə

ə ə ə 11 ə’

ə ə ə

ə ə ə

ə ə ə ə ö ə

ə ə ə ə ə ə 1 ə ü ə ə

ü ə ə ə ə ğ ğ 1 1 Ə ə

ü ə ə 1 ə ə ə ə 1 1 üçü ə ə

1

ə ə ə ç ə ə ə 1 1 1 ə

1 1 1 ə ə ğ ə ə ə ə

ə ə 1 ə ə 1 1 ə ə ə ə ___ ə

ə ə 1 ə 1 1 ə ə ə ə 1 1

ə ə ə 1 1 ə’ Ə ə – 1 1 ə –

1

ə ə 1 ə ğ ğ

1 ə 1 ə’ 1 1 1 Ə ə –

ğ ə – 1

Şərti və şərtsiz kecid operatoru.

DƏYİŞƏNLƏR - onların tipləri və dəyişənlər ğzərində hesab əməlləri ilə tanış olduq. Bu paraqrafda isə ə 1 ış 1 ə ə

ə ə ış 1

ş 1 ə 1 şə ə 1 1 1

ə ə

ə ə ə əş üçü ə 1 şə ə 1 1 ə

ə

ə ə 1 ə ə ə ə ə ə ə

ə ə

Şə ı

ış ğı **Şə ə** Şə

ə Ə ə

ə ə ə ə 1

üçü

ə ə ə ə şə ə

ə

Şə ə 1 ə ə şə ə 1 1

ə ə ə ş ə ə

ə ə ə ə Ə ə ə ə ə ə

ş ğı 1

Şə ş ğı 1

şə

ə ə ə ə

ə ə ə

ə ə ə ə ə ə ə ə Ə ə ə ə ə

ı

Ə ə ə şə 1 1 ə ə ğ 1 ə ə ə

ə

– ə 1 Ə ə ə ə 1 ə ə ə

ə

Ə ə ş ə 1 üçü ə şə

ə ə ə ə

şə ə 1 1 1 ə ə əş ə ə

ş ğı 1

şə şə şə

Ə ə 1 1 ə 1 üçü **şə şə şə** ə

ə ə ə ə ə şə ə 1 1 1 ə

ə əş ə ə

ş ğı 1

ş ə şə şə

ə ə ə ə ə 1 ə ə

şə ə ə ə 1

şə ə 1 ə ə 1 ə ə

ə ə Şə ə ə

1 ə ə ə ə ə ə ə ə ə ə ə

ə ə ə ə ə

ü ə ı

şə ə ə ə ə 1 ə

ə ş ğı 1 ə 1 ə

ə

ə ə ə

ə ə

ə ə

ə

1 1 ə ə ə ə ə ə

ış ə ə 1 1 ə ə

C ış İ ə ə ə ə ə ğ ə ə

ə

ə 1 ə ə üçü ə ə ə ə

ə 1 1 ə ə ə ə Ə ə ə

ə ə ə Ə ə ə ə 1 1 ə üçü

ə ə 1 ığı ə ə ə ə Ə ə ə ə ə

ə ə 1 1 ə ə ə ə ə Şə

ə

ə ə ə ə 1 1 1 ş ğı 1

Switch, Do dövr operatoru.

switch operatoru

Ə ə ə ə 1 ə ə şə ə 1 1 1

ə şə ə ə ə 1 ə

ə ə ə ə ə ə

ş ğı 1

ə şə

ə

ə ə ə şə ə ə

ə ə ə

ə

ə ə ə şə ə ə

ə ə

ı şə ə ə ə ə

ə ə ə

ş ə ə ə 1 ş ğ1 ə

ş1 1 ə ə 1 ə ə ə ə ə ə ə

ə ə 1 ə ə ə ə

ş 1 1 ə ə ə ə ə ə

ə ğ 1 ə ə

ə ə ə ə ə şə ə ə ə ə

ə ə ə Ə ə ə ə ə

ə ə ə ə ə ə ə ə ə ə ə ə ə

ə

1 ə ə

şı

continue və break

ş ə ə 1 ə ə üçü ə ə

ə ə ə üçü ə ə ə 1

ə ə ə üçün ğ ə ğ ə ə ə ə

ə ş 1 1

ğ 1 ş ğı ə ə Şə

ğ 1 ə ş ə

ğ 1 ş ğı ə ə Şə ə

ğ 1 ə ş ə ə 1 1 ə ə ə ə

ğ ı ş ğı ə

ğ ə ə ə ə ğ ə ə ə

şə ə 1 1 ə ə ə ə ə

ə ə ə ə ə ə ş ə ə ə ə ə ə 1 1

ğ 1 ə ə ə ə ə ə ə ə

ə ə ə üçün ə ə ğ ə şə ə ə

ə ə ş ğı ə ə ə ş 1

ə Şə

ə şə 1 ə ə ə 1 1

ə ə əş ğ 1 ə 1 ə ə ə ə

ə

ğ ı ə ş ə

ə ə ə 1 1 ə ə ə ə ə ğ 1

ə ş ə ğ 1 ə ə ə ə ə ə ə

ə ə ə ə ış

ış ə ə ə

ç ə ə

ə ə ə üçü

ə ə ə ə ə ə ə 1 ə ə ə

ə ğ ə şə ə ğ ş ğı

ə ə ə ə Ə ə ğ 1 ə ə ə ə

ə ş

şə ə ə ə ə ğ 1 ə

ə ə ə 1 üçü ə ə

ə ğ 1 şə ə ş ğı 1

ə ö ü ə ə 1 1 1 ə 1

ə ə ə ə ə ə

ə ə ə ş ğı 1

ə

ış ə ə ə ə ə ə ə ə

ə ə

ə ə ə ə ə ə ə ə ə ə

üçün ğ ş ğı ə ə ə ə şə

ə ə ə ə ə üçü ğ 1 ə ə ə ə ə

1 1 ə ə ə ə ə 1 ğ 1 ə

ə ş ğı 1

ış ə ə ə ə ə ə ə

ə ə ə ə

ə

ə ə

ı ə ə şə ə ə ğ ə ə

ə üçün ə ə ə ğ ə ə

ə ə ə ə ə ə 1 1 1 ə ə ə ğ ə

ə ə ə 1 ə ə ə ə ə ə ə ə ə ə ə

ə ə 1 – ə ə ə 1 ə ə ə ə

ə ə ə ə ə ə ə ə ə ə 1

ğ ə ə ə 1

şə

ə ə ə ş ğı 1

ə ş Şə

ə ə ə şə ə ə ə ğ ə 1 ə

ə

ə ə ə ş Şə ə ə şə ə

c ış ə ə ə ş ğı 1 ş ə ə

ə İ ə ə ə ə ə Ə

ə ə ğ ə İ ə ə ə ə

ə 1

ə ə ə ə ə ə ə 1 ə ə əş ə üçün

ə ə ə ə ə ə ə ə ə ə ə ə

ə ə ə ə ə ə ə ə ə ə ə ə

ə ə ə ə ə ə ə 1 1 İ ə

ə ə üçü

1 ə üçü ə şə ə ə ə Ə ə ə

ə ə ə ə ə ə 1 ə ə ə ə ə

ə

ş şə ə ə ə ə ə ə

ş ğı 1

do while operatoru while operatoruna analojidir, sintaksis aşağıdakı kimidir:

```
do{
Proqram Kodu;
} while ( Dцvrьn Başa Зatma Şərti );
```

1. Continue operatoru, return operatoru

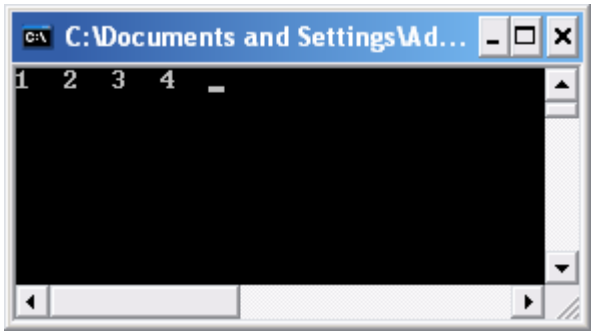
break **və** ***continue*** **operatoru**

Bildiyimiz kimi ***break*** operatoru *switch* strukturundan çıxışı təmin edir. Bundan əlavə dövri strukturların daxilində də *break* operatorundan istifadə edilir. Bu operator icra olunduqda dövr dərhal başa çatır, yəni *break* operatorundan istifadə etməklə müəyyən şərt daxilində dövrü sonlandırmaq mümkündür.

Misal 1.

```
#include<iostream.h>
 // #include<stdio.h>
#include<conio.h>
main()
{
 for(int i=1;i<=10;i++)
 {
 if(i==5)
 break;
 cout<<i<<"  ";
// printf("%d  ",i);
 }
 getch();
 return 0;
}
```

Alınmış nəticə:

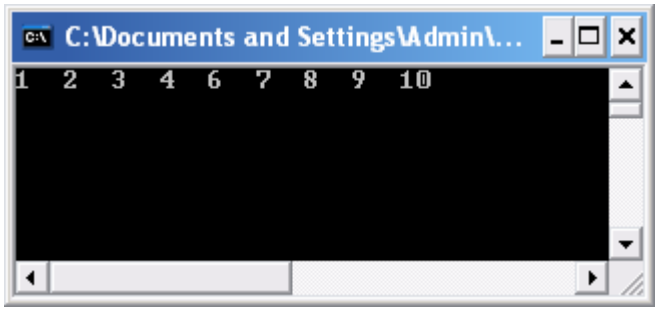


continue – dövrün cari iterasiyasını qurtarmaq və dövrün növbəti iterasiyasına keçid üçün təyin olunub. Daha dəqiq desək, *continue* operatoru yerinə yetirildikdə dövr daxilində *continue*-dən sonra gələn əməliyyatlar icra olunmadan dövr yeni tsiklə keçir. *continue*-nin vasitəsilə dövrün daxilində istənilən nöqtədə dövrü saxlamaq və idarəni *for* və ya *while* operatorlarındakı dövrün davam etmə şərtinin yoxlanmasına vermək olar. Beləliklə də hesabatı dövrün növbəti qiymətləri üçün aparmaq olar.

Misal 2.

```
#include<iostream.h>   // #include<stdio.h>
#include<conio.h>
main()
{
 for(int i=1;i<=10;i++)
 {
 if(i==5)
 continue;
 cout<<i<<"  ";
 // printf("%d  ",i);
 }
 getch();
 return 0;
}
```

Alınmış nəticə:

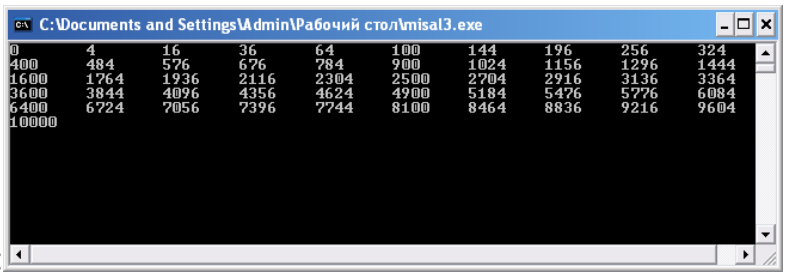


Misal 3. [0,100] aralığında cüt ədədlərin kvadratlarını ekrana verməli.

```
#include<iostream.h>  // include<stdio.h>
#include<conio.h>
main()
{
int n;
for(n=0; n<=100; n++){
if (n%2) continue;
cout<<n*n<<"\t";
//printf("%d", n*n);
}
getch();
return 0;
}
```

Burada *n* tək olduqda *n%2=1* olur və *if* operatorundakı *continue* operatorunun köməyi ilə dövr parametrinin yoxlanılmasına ötürülür. Yəni çıxış funksiyası icra edilmir.

Alınmış nəticə:



Bilirik ki, Si proqramlar adətən funksiyalar ardıcıllığından təşkil olunur. Hər bir funksiya isə return (əsas proqrama qayıdış) operato-ru ilə qurtara bilər. Yazılışı

return [(<ifadə>)] ; kimidir.

Bu operator idarəni əsas proqramda funksiya çağrılan nöktə-dən sonrakı operatora qaytarır. Əgər operatorun yazılışında<ifadə> verilərsə <ifadə>- nin qiyməti cağrılan funksiyanın qiyməti kimi əsas proqrama qaytarılır. Əgər funksiya return operatoru ilə qurtarmazsa idarə avtomatik olaraq əsas proqrama qaytarılır və qaytarılan qiymət qeyri müəyyən olur. Beləliklə bu operator ilə funsiyadan iki cür keçid təşkil edilir: nəticəni qaytarmaqlavə ya nəticəni qaytarmadan.

Proqramların layihələndirilməsi

Məsələnin qoyuluşu

Proqramların layihələndirməsinin ən vacib mərhələlərindən biri məsələnin qoyuluşudur. Əwəlcə proqram təbii dildə ifadə edilir. Əgər iri layihə yerinə yetirilirsə, onda proqrama aid bütün tələblər yazılı müqavilədə öz əksini tapmalıdır.

Qoyuluşu yaxşı və pis olan məsələləri fərqləndirmək olar. Yaxşı qoyulmuş məsələlərdə ilkin verilənlər aydın görünür, verilənlər arasında olan bütün əlaqələr təyin olunub. Bu da birmənalı nəticəyə gətirib çıxardır.

Əgər birmənalı həll üçün ilkin venlənlərin sayı kifayət deyilsə, yəni ilkin veriiənlər və nəticə arasında əlaqələr tam aydın göstərilməyibsə, onda məsələnin qoyuluşu o qədər yaxşı olmur.

Məsələn, "Uşaq məktəbə getməlidir. Məktəbə çatmaq üçün ona nə qədər vaxt lazım olacaqdır?" məsələsi yaxşı qoyulmamış məsələlərdən biridir. Bu məsələnin həlli üçün kifayət qədər məlumat verilməyib. Amma, əgər bu məsələnin qoyuluşunda məktəbə qədər məsafə və şagirdin sürəti göstərilsə idi, onda onu asanlıqla həll etmək olardı.

Verilənlər modelinin yaradılması

Praktiki proqramlaşdırmada əsas problemlərdən biri - məsələnin formal modelinin (riyazi və məntiq dillərində), verilənlərin lazımı strukturun (dəyişənlər, massivlər, strukturiar) və əlaqələrin qurulmasıdır. Proqramlarda ən ciddi səhvlər verilənlər modelinin düzgün qurulmaması ilə bağlıdır.

Alqoritmin hazırlanması

Bu mərhələdə alqoritmi seçmək və ya yenidən hazırlamaq lazımdır.

Məsələnin həllinə gətirib çıxaran müəyyən edilmiş sonlu əməliyyatlar ardıcıllığı alqoritm adlanır.

Alqoritm qabaqcadan seçilmiş verilənlər modelini nəzərə almalıdır. Bəzən əwəlki mərhələyə qayıtmaq lazım gəlir, çünki seçilmiş alqoritmi istifadə etmək üçün başqa cür verilənləri istifadə edilməlidir. Məsələn, massiviərin əvəzinə strukturiardan istifadə daha

Proqramın hazırianması

Tapşırığı başa düşmək üçün o. kompüterə yaxın olan dillərdən birində tərtib olmalıdır. yəni alqoritm əsasında proqram yazılmalıdır.

Proqramlaşdırma dilində yazılmış aİqoritm proqram adlanır. Ço* vaxt kompüterə yazılmış əmrlərin (jnsteulçgyajann] yığırr.ına da proqram deyılır.

Proqramı yazarkən proqramİaşdırma dilini seçmək vscibdir. Peşəksr. yəni yüksək səviyyəli proqrsmlsn. adətən. C və ya C++ dilində tərtb edirlər.

Proqramın düzənnəməsi

| Proqramda səhvlərin axtarışı və düzəldilməsi düzənnəmə (ing. debugging) adLanır.

Proqrsmlsnn düzənnəməsl üçün debugger adLarlan xüsusl proqrsmlsrdsn istifsdə edilir. Debuggerin köməyi ilə aşağıdakıİarı yerinə yetirmək oLar

- proqrsmı addım-addım (hər əmrdən sonrs daysnaraq) yerinə yetirmək:
- proqramda kəsilmə nöqtələrini qoymsq:
- yaddaşa yszılmış dəyişənləri və prosessor registrlərini müşshidə etmək və qiymətlərini dəyişmək:
- düzənnəmə Zamanı profiler adİanan proqramdan ds istifsdə etmək oİar. Bu proqrsm Vasitəsi ilə proqram hissələrinin neçə Vaxta yerinə yetirilməsini təyin etmək və ləng işləyən hissələri optimaİİaşdınriaq olsr.

Sənədlərin hazırianması

Proqramls bağlı sənədlər düzgün tərtib olunmalıdır. Adətən. istifadəçi üçün təlimat (fser *manuaf).* bəzən proqrsmçılsr üçün təlimst (P/ogrammer's *manual).* aİqoritmin və proqram xüsusiyyətlərinin ətraflı təsviri haZirİanır. Bu sənədlər istifsdəçiyə aydın və Sadə dildə təqdim olunmslıdır.

Proqramın sınaqdan keçirilməsi

Xüsusi hazırlanmış insanİar (tester) tərəfindən proqrsmın yoxİanılmaSi onun sınaqdan keçirilməsi adİanır.

Sınağın məqsədi hazırİanma zsmsnı proqramda aşkar olunmsysn səhvləri üzə çıxsrtmsqdır. Peşəkar proqrsmLsrın sınsğı iki mərhələdən ibsrətolur

- alfa-testing - istehsalçı firmanın əməkdaşlsrı tərəfindən həyata keçirilən sınsq:
- beta-testing - digər firmalsr və təşkiİatlarda proqramın ilkin versiyssının sınağı; çox vaxt proqramın beta-versiyaİarı İntemet Vasitəsi ilə sərbəst ysyımlsnır.

Şərti olsrsq, proqrsmı o Vaxt düz hessb etmək olsr ki. nə vsxt seçilmiş ilkin test verilənlər üçün onun yerinə yetirilməsi düz nəticəyə gətirib çıxsrdır.

Proqrsmın sınsğı üçün nəticələri qabaqcadan məlum olan etaİon misaİları hszıiismsq lazımdır.

Müşayiət etmə

Sifarişçiyə təhvil veriləndən sonra proqramda səhvlərin düzəldilməsi və istifadəçilər üçün məsləhətlərin verilməsi proqramın müşayiət etməsi adİanır.

"Aşağıdan ynxanya doğru" proqranılaşdırma

Qabaqİarbuyanaşmadantez-tezistifadəedilirdi. Proqramın hazırlanmssı ən sadə prosedura və funksiyaİardan baŞİayır. Aşsğı səviyyəli (sadə) prosedураİan yazaraq. onlardan daha iri və mürəkkəb prosedураİarı və funksiyslsrı qururuq və s. (kiçik kublsrdsn evin tikintisinə oxşsyır). Son məqsəd - kiçik kublsrdsn bütün proqramın tam yığılmasıdır. Üstunlüklər:

- proqramı "sıfırdan" hazirlamaq asandır.
- "Sadədən-mürəkkəbə" üsulundan istifsdə edərək dsha effektiv prosedураİarı yazmaq oİar.

Çatışmayan cəhətlər:

- Sadə prosedураİann hszıiisnmssı Zamsnı onlsn bir növ məsələnin qoyuluşu ilə bağlsmsq Iszımdır.

- elə o)a bilər ki. soruncu mərhə<ədə hər hsnsı kubİar (proseduraİar) çstışmasın:
- proqram çox dolaşıq olur.

Massivlərin elanı, birölçülü massivlərinə müraciət, cərgə ilə ünvan dəyişənləri arasında əlaqə, cərgələrin funksiyaya parametr kimi ötürülməsi.

Eyni ada malik nizamlanmış kəmiyyətlər yığımı massiv adlanır. Daha dəqiq desək, massivlər elementlərinin sayına və strukturuna görə nizamlanmış və nömrələnmiş bircinsli verilənlər yığımıdır.

İndekslərin sayı massivin ölçüsünü göstərir. Massivlər bir ölçülü (vektorlar), iki ölçülü (matrislər) və çox ölçülü olurlar.

Bir ölçülü massiv vektor, iki ölçülü massiv isə matris adlanır. Massivin elementləri massiv də ola bilər. Bu zaman massivə çox ölçülü massiv deyilir.

Bir ölçülü massiv - eyni bir addan istifadə edən bir neçə eyni tipli dəyişəndən ibarət olur və hər bir dəyişənə müraciət onun indeksi ilə, yəni sıra nömrəsi ilə həyata keçirilir.

Cədvəldə yerləşən məlumatların emalı üçün bütün müasir proqramlaşdırma dillərində iki ölçülü massivlərdən istifadə edilir. İki ölçülü massiv sətr və stunlardan ibarət olur. Hər element iki indekslə, yerləşdiyi sətir və sütunun sıra nömrəsi ilə təyin olunur.

Massivin elanı

Massivə ad verərkən identifikatordan istifadə olunur. Massivə daxil olan elementlərin tipi massivin tipini müəyyən edir. Massivlər müxtəlif tipli ola bilərlər.

Massivdən istifadə etmək üçün onu elan etmək, yəni ona yaddaşda yer ayırmaq lazımdır.

Bir ölçülü massivlər aşağıdakı formada elan olunur:

Elementlərin_tipi massivin_adı[elementlərin_sayı];

Məsələn, 12 elementdən ibarət tam tipli bir ölçülü ədədi massivi aşağıdakı şəkildə elan etmək olar:

int a[12];

İki ölçülü massivlər aşağıdakı formada elan olunur:

Elementləriin_ tipi massivin_adı [ölçü1] [ölçü2];

Burada *ölçü1* – sətirlərin sayı, *ölçü2* – sütunların sayıdır.

3 sətr və 4 sütundan ibarət həqiqi tipli iki ölçülü ədədi massiv isə aşağıdakı şəkildə elan olunur:

float b[3] [4];

Proqramların universiallığını artırmaq üçün massivin ölçüsünü konstant vasitəsi ilə təyin etmək məqsədə uyğundur. Bu halda proqramı başqa ölçülü massiv üçün yazmaqdan ötrü yalnız konstantın qiymətini dəyişmək kifayət edir. Məsələn,

const int n=3, m=4, k=12;

int a[k];

float b[n][m];

Qeyd edək ki, massivlərin elementlərinin sayı indeksin təyin olunduğu intervalı aşmamalıdır. Əks halda , proqram yerinə yetirilərkən səhv aşkar olunaçaq.

Bir ölçülü massivin elementləri yaddaşda ardıcıl olaraq yerləşir. İki ölçülü massivin elementləri isə yaddaşda sətir-sətir, yəni sağ indeksin artmasına uyğun olaraq ardıcıl yerləşirlər.

Massivin elanında elementlərə başlanğıc qiymət də vermək mümkündür. Məsələn, tam tipli bir ölçülü *a[12]* ədədi massivinin elementlərinə aşkar şəkildə başlanğıc qiymətləri vermək üçün aşağıdakı yazılışdan istifadə olunur:

int a[12]={32, 14, 28, 96, 87, 14, 63, 48, 12, 11, 10, 19};

Əgər elandakı qiymətlərin sayı massivin elementlərinin sayından az olarsa, qalan elementlərə avtomatik olaraq sıfır başlanğıc qiyməti verilir. Deməli,

int a[12]={0};

elanında *a* massivinin bütün elementlərinin ilkin qiyməti sıfır olacaq.

Həqiqi tipli iki ölçülü *b[3][4]* ədədi massivinin elementlərinə aşkar şəkildə başlanğıc qiymətləri vermək üçün aşağıdakı yazılışlardan biri istifadə oluna bilər.

float b[3][4]={{1,2,3,4}, {5,6,7,8},{9,0,0,0}};

float b[3][4]={{1,2,3,4}, {5,6,7,8},{9}};

float b[3][4]={1,2,3,4, 5,6,7,8,9};

Massivin elementlərinə müraciət olunarkən massivin adı və elementin indeksindən istifadə edilir. Bunun üçün massivin adını və kvadrat mötərizənin içərisində indeksi göstərmək lazımdır. İndekc istənilən tam ədəd və ya tam qiymət alan ifadə ola bilər.

C/C++ dilində massivlərin indeksləşdirilməsi sıfırdan başlanır. Məsələn, *a* vektorunun birinci elementinə müraciət etmək üçün *a[0]*, ikinci elementinə müraciət etmək üçün *a[1]*, on ikinci elementinə müraciət etmək üçün isə *a[11]* yazılışından istifadə edilir. *b[0][2]* yazılışı isə *b* matrisinin birinci sətirlə üçüncü sütununun kəsişməsində yerləşən elementə müraciəti təmin edir.

Massivlərin daxil və xaric edilməsi

Massiv elan olunduqdan sonra massivin elementləri təyin edilməlidir, yəni massivin hər bir elementinə onun baza tipinə uyğun bir qiymət və ya ifadə mənimsədilməlidir.

Massivin daxil və ya xaric edilməsi üçün dövr operatorundan istifadə olunur.

Massivi yaddaşa daxil etmək üçün onun hər bir elementi *cin* obyekti və ya *scanf* funksiyası vasitəsilə oxunmalıdır. Sadə proqramlarda massivi klaviaturadan daxil edirlər. Burada elementlərin sayı az olur. Massivi ekrana çıxarmaq üçün isə *cout* obyekti və ya *printf* funksiyasından istifadə olunur.

Bir-neçə nümunəni nəzərdən keçirək:

a) bir ölçülü massivin daxil edilməsi

```
float a[10];
int i;
for (i=0; i<10; i++)
cin>>a[i];
// scanf("%f", a[i]);
```

b) iki ölçülü massivin daxil edilməsi (sətirlər üzrə)

```
float b[3][4];
int i, j;
 for (i=0; i<3; i++)
 for (j=0; j<4; j++)
  cin>>b[i][j];
 // scanf (“%f”, b[i][j]);
```

c) bir ölçülü massivin xaric edilməsi

```
 float a[10];
 int i;
 for (i=0; i<10; i++)
  printf(“%8.2 f ”, a[i]);
```

d) iki ölçülü massivin xaric edilməsi (sətirlər üzrə)

```
 int b[3][4];
 int i, j;
 for (i=0;i<3;i++){
 for (j=0;j<4;j++)
 printf (“%10d”, b[i][j]);
 printf(“\n”);}
```

Dinamik və Adi dəyişənlər

Gələcəyin güclü proqramcıları ilə tələbələrin ayrıldığı yerə catdıq.
Proqramda istifadə olunan dəyişənlər yaddaşda yer ayrılmasına görə iki cür olur, adi və dinamik . Bizim indiyə qədər istifadə etdiyimiz dəyişənlər hamısı adi dəyişənlərdir. Aşağıda onların müqaisəsi verilir.
Adi dəyişənlər :
ancaq proqramın əvvəlində onlara yer ayrılır,proqramın icrası boyu onlara ayrılan yer olduğu kimi qalır və bu yer dəyişəndən geri alınıb hansısa başqa məqsəd ücün istifadə oluna bilməz, proqramın icrası boyu yaddaşda eyni bir ünvana istinad edirlər, bu dəyişənlərin istinad etdiyi ünvanı dəyişdirmək olmaz.
Dinamik dəyişənlər :
Dinamik dəyişənləri proqramın icrasının istənilən anında yaratmaq olar,onlara ayrılmış yaddaş proqramın icrasının istənilən anında geri alıb həmin yeri istənilən digər məqsəd ücün istifadə etmək olar,dinamik dəyişənlərin yaddaşda istinad etdikləri ünvanı istənilən digər ünvana dəyişdirmək olar, hətta digər proqramın və ya nüvənin yaddaş sahəsinə.Ancaq buna etmək istədiyimiz ilk cəhddə nüvə proqramımızı təmamilə söndürər(başqa proqramların məlumatlarına icazəsiz müraciət etmək olmaz).
Ünvan dəyişənlərinin özəlliyi odur ki, onlar özlərində məlumat olaraq bizim ücün əhəmmiyətli olan məlumatın ünvanın saxlayırlar. Əgər biz bu ünvanı dəyişsək onda onlar ayrı məlumata istinad edəcəklər. Bu C/C++ dillərinə xarakterik olan, proqramcıya yaddaşla istədiyi kimi manipulyasiya etmə imkanı yaradan güclü vasitələrdən biridir.
Proqramda hər hansı tipdən ünvan dəyişəni elan etmək ücün aşağıdakı sintaksisdən istifadə olunur:

tip *dəyişən;

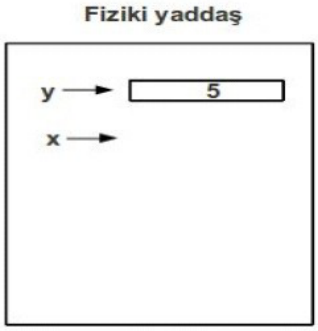
Tam tipli hər-hansı ünvan dəyişəni elan edək:

int *x;

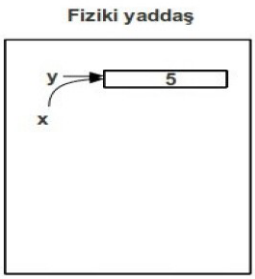
Göründüyü kimi bunun adi dəyişən elan etmək qaydasından (**int x;**) fərqi ancaq dəyişən adının əvvəlində * - ulduz simvolunun olmasıdır.
Bu zaman yaddaşın vəziyyəti belədir(şəkil 5):

ti:

İndi mən x ünvan dəyişənini istənilən yaddaş ünvanına və ya istənilən dəyişənin yaddaş sahəsinə mənimsədə bilərəm və ya yaddaşda dinamik şəkildə əlavə yer ayıra və həmin yerə mənimsədə bilərəm.
Misal ücün gəlin adi int tipli y dəyişəni elan edək(**int y;**) və onun yaddaş sahəsinə 5 qiyməti yazaq(**y = 5;**).
Yaddaşın vəziyyəti:



İndi mən x -i y -in yaddaş sahəsinə mənimsədə bilərəm.
Bunun ücün **&** ünvan operatorundan istifadə edəcəm.
& opratoru istənilən dəyişənin və funksiyanın ünvanını almaq ücün istifadə olunur.
x -i y -in ünvanına məminsətmək ücün sadəcə olaraq yazırıq:
x = &y ;
Yaddaşın vəziyyəti:



Əgər bizə ünvan dəyişəninin istinad etdiyi ünvana müraciət etmək lazımdırsa onda sadəcə ünvan dəyişəninin adından istifadə edirik. Əgər biz həmin ünvanda yerləşən məlumata müraciət etmək istəyiriksə bu zaman dəyişənin əvvəlinə elanda oluğu kimi * simvolu artırırıq.
Gəlin bu dediklərimizi proqram nümunəsində test edək.

```
//prg_4_1.cpp
#include<iostream>
int main(int argc, char *argv[])
{ int *
x; // tam tipli unvan deyisheni
int y;
y=5;
x=&y; // & unvan operatorudur
// y -in qiymetini cap edek
```

```
std::cout<<"y-in qiymeti = "<<y;
// x-in qiymetini cap edek (aha y-in unvani)
std::cout<<"¥n x-in qiymeti (y-in unvani) "<<x;
// x unvanında yerləşən məlumatı cap edək
std::cout<<"¥n x-in istinad etdiyi melumat "<<*x;
// x-in istinad etdiyi məlumatı deyishek
*x=67;
// y -in qiymetini cap edek
std::cout<<"y-in yeni qiymeti = "<<y;
return 0;
}
```

Simvol və sətirlərin sintaksisi

Hərflər, durğu işarələri və idarəedici simvollar C/C++ dilində simvollar adlanır. Simvollar adətən bir-birinin ardınca yazılır və sözlər, cümlələr və s. əmələ gətirirlər.

Bir simvol səkkiz bitlə kodlaşdırılır və o, *char* dəyişənində saxlanıla bilər. Bildiyimiz kimi, *char* tipi işarələr çoxluğundan ibarətdir. Bu çoxluq 256 işarədən ibarət olub ASCII işarələrindən təşkil olunmuşdur. Simvollar təkqat dırnaq işarəsi daxilində yazılır və aşağıdakı kimi təsvir olunur:

unsigned char x;

x='a' ;

Proqramın bu fraqmentinin yerinə yetirilməsindən sonra x dəyişənində latın əlifbasının kiçik *a* hərfi kodlaşdırılan ədəd yerləşəcək.

Bəzi simvollar görünməyəndirlər, onların yazısı üçün dırnaq işarəsi daxilində hərflər ardıcıllığı istifadə olunur. Məsələn, *'\n'* – sətrin keçidi, *'\r'* – karetkanın qaytarılması.

C/C++ dilində xüsusi sətir tip verilənlər yoxdur və sətirlər üçün simvol (*char*) tipli massivlər istifadə olunur. Sətir – ikiqat dırnaq işarəsi daxilinə alınmış simvollar ardıcıllığıdır.

Sətirlərlə işləyən zaman diqqət verilməsi lazım gələn məsələlərdən biri də sətirin sonu məsələsidir. Sətirlərin sonunu bildirmək üçün sıfır kodlu simvoldan istifadə olunur. Sıfır kodlu simvolun təsviri yoxdur. Proqramda onu *'\0'* kimi yazırlar. Sətir funksiyaları bu simvola rast gəldikdə sətrin yerdə qalan hissəsini inkar edirlər.

1.4 Sətirlərin elanı və başlanğıc qiymətlərin verilməsi

Sətirlər üçün simvol tipli massivlərdən istifadə olunduğuna görə sətirlər massiv kimi elan olunur. Məsələn,

char s[20]; //20 simvoldan ibarət sətir;

char d[ ];//simvollarının sayı məlum olmayan sətir; Lakin massivlərdən fərqli olaraq sətir *null '\0'* simvolu ilə bitməlidir. Bu səbəbdən əgər simvollar massivi sətir kimi istifadə olunacaqsa, onun elanı üçün 1 bayt artıq yer verilməlidir. Qeyd etmək lazımdır ki, massivlər ölçüsu olmadan elan oluna bilər. Sətri massivə salmaq üçün

char a[6]= "Hello";

yazmaq kifayətdir. Bu sətrə rast gəldikdə kompilyator özü altı simvoldan ibarət massiv - beş hərf və yekunlaşdırıcı *null '\0'* simvolu yaradır.

Əgər proqramın icrası zamanı sətir dəyişməyəcəksə, onda onu sabit kimi elan etmək olar:

const char a[]="Hello";

Sətirlərə qiyməti üç üsulla vermək mümkündür:

inisializasiya //elan vaxtı
mənsubetmə // icra vaxtı
daxiletmə
1-ci və 3-cü üsulda *'\0'* simvolu avtomatik əlavə olunur, 2-ci üsulda proqramçının özü onu təmin etməlidir.

Nümunə:

char x1[6];

x1[0]='H';x1[1]= 'E';x1[2]= 'L';x1[3]= 'L';

x1[5]= 'O';x1[5]= '\0';

char m[3][25]={ "Misal","Kamil","685"};

*char *n[]={"one","two","three","four","five"};*

*// char **n;*

Burada *n[1]* elementinə *"two"* sətri, *n[1][2]* elementinə isə *'o'* simvolu uyğundur.

Standart daxiletmə və xaricetmə

C dilində sətrlərin daxil və xaric edilməsi üçün *scanf* və *printf* funksiyalarında xüsusi "***%s***" formatından istifadə olunur:

#include<stdio.h>

#include<conio.h>

main(){

char name[20];

printf("What is your name?");

scanf("%s", name);

printf("Hello, %s",name);

getch();

return 0;}

scanf funksiyasının bir çatışmayan cəhəti var: birinci boşluğa rast gələndə daxiletmə dayanır. Əgər sətrin bütövlükdə daxil edilməsi tələb edilirsə, *scanf* funksiyası əvəzinə ***gets*** funksiyasından istifadə edilməlidir:
gets(s);

Sətrin ekrana çıxarılması üçün *printf*-dən əlavə ***puts*** funksiyasından istifadə etmək olar. Bu funksiya sətri ekrana çıxardıqdan sonra kursoru növbəti sətrin əvvəlinə gətirir.

Aşağıdakı

gets, puts funksiyaları ilə işləmək üçün proqrama *stdio.h* başlıq faylını daxil etmək lazımdır.

Funksiya nədir

Bir çox məsələlərin həllində eyni tipli, amma müxtəlif verilənlərlə icra olunan hissələrə tez-tez rast gəlmək olur. Təkrarlanmanı aradan qaldırmaq, proqramı daha aydın və sadə etmək üçün bütün proqramlaşdırma dillərində belə hissələri ayrı və müstəqil alt proqramlara çevirən vasitələr mövcuddur. Belə xüsusi proqramlar iki cür olur: prosedura və funksiya. Başqa dillərdən fərqli olaraq C/C++ dilində ancaq funksiyalardan istifadə olunur. Funksiya – hər hansı məsələni yerinə yetirən operator-lar küllisidir. İndiyə qədər bizim proqramlarda yalnız bir funksiyadan istifadə olunurdu (*main*).

Funksiyalar proqramın hər hansı bir hissəsinə ad verməklə bu hissəyə proqramın istənilən yerindən, digər funksiyalardan və eləcə də digər proqramlardan müraciət etməyə imkan verir. Bu hissəyə funksiyanın mətni, bu hissəyə verilən ada isə funksiyanın adı deyilir.

Proqramda birdən cox funksiyanın olması məcburi deyil. Lakin mürəkkəb proqramların yaradılmasında funksiyasız, demək olar ki, keçinmək olmaz. Belə yanaşmanın bir neçə üstün cəhəti var:

- proqramlar hissə-hissə yazıla bilir;
- hazır hissələr başğa proqramlarda istifadə oluna bilir;
- proqramın ümumi yaddaşı azalır;
- proqram daha aydın və strukturlaşmış olur;

Hər bir funksiya ilə işləmək üçün onu elan etmək, təyin etmək və ona müraciət etmək lazımdır.

Funksiyanın elanı

Funksiyaları əsas proqramdan əvvəl elan etmək lazımdır. C/C++ dilində funksiya aşağıdakı kimi elan olunur:

nəticənin_tipi funksiyanın_adı (tip1 arqument1,tip2 arqument2, ...);

Burada *nəticənin_tipi* funksiyanın qaytaracağı nəticənin tipini göstərir. Əgər funksiya heç bir nəticə qaytarmırsa, onda *nəticənin_tipi* olaraq ***void*** yazılır.

funksiyanın_adı operator adları ilə üst-üstə düşməməlidir. Funksiyalara ad verərkən identifikatorlardan istifadə olunur.

Funksiyanın adından sonra mötərizə daxilində funksiyanın qəbul edəcəyi arqumentlərin siyahısı verilir. Arqumentlər bir-birindən vergüllə ayrılır. Əgər funksiya heç bir parametr qəbul etmirsə, mötərizə daxilində *void* yazılır.

Funksiyanın elanında arqumentlərə verilən adlar heç bir əhəmiyyət daşımır və onlar buraxıla bilər. Burada əsasən arqumentlərin tipi önəmlidir.

Məsələn:

int perimetr(int a, int b);

burada *int* tipli nəticə qaytaran və *int* tipli iki arqument qəbul edən *perimetr* funksiyası elan olunub. Biz bunu aşağıdakı kimi də yaza bilərik:

int perimetr(int , int);

burada arqumentlərin adları göstərilməmişdir.

Funksiyanı elan etməklə, daha dəqiq desək, funksiyanın prototipini yaratmaqla biz kompilyatora funksiya (ad, tip, arqumentlər) barəsində məlumat vermiş oluruq. Prototip bütün funksiyaların təyinindən yüksəkdə dayandığına görə kompilyator kodu yuxarıdan aşağıya keçərkən hər bir funksiyanın qarşısında dəqiq prototip görür.

Funksiyanın təyin olunması

Funksiyalar adətən əsas proqramdan sonra təyin edilir. Funksiyanı təyin etməklə, yəni funksiyanın mətn kodunu tərtib etməklə biz onun görəcəyi işi proqramlaşdırmış oluruq. Bunun üçün aşağıdakı qaydadan istifadə olunur:

nəticənin_tipi funksiyanın_adı (tip1 arqument1,tip2 arqument2, ...){
proqramın kodu
***return** nəticə;*
}

Funksiyanın başlığı nəticənin tipindən, funksiyanın adından və parametrlər siyahısından ibarət olur. Funksiyanın başlığında yazılmış parametrlər formal parametrlər adlanır. Bu o deməkdir ki, onlara yalnız funksiya daxilində müraciət etmək olar.

Qeyd edək ki, funksiyanın başlığı prototipə oxşayır, əsas fərq ondadır ki, prototip operatordur və ondan sonra həmişə nöqtə vergül yazılır.

Funksiyanın başlığından sonra ***{*** mötərizəsi ilə funksiyanın gövdəsi başlayır və bağlayıcı ***}*** mötərizəsi ilə qurtarır. Qeyd edək ki, funksiyanın gövdəsində başqa mötərizələr də iştirak edə bilər. Funksiyanın gövdəsində funksiyanın proqram kodu yerləşdirilir. Burada hətta digər funksiyalara müraciət də oluna bilər. Əgər funksiya *void* tipdəndirsə, yəni heç bir qiymət qaytarmırsa, təbii ki, funksiyanın gövdəsində *return* operatoru iştirak etmir.

Funksiyanın gövdəsi və başlığı birlikdə *funksiyanın təyini* adlanır.

Yuxarıda elan etdiyimiz *perimetr* funksiyasının proqram kodunu nəzərdən keçirək.

```
int perimetr (int a, int b)
{
int p;
p=2*(a+b);
return p;
}
```

Burada funksiyanın daxilində *int* tipli *p* dəyişəni elan olunub. Daha sonra *p* dəyişəninə funksiyanın arqumentlərinin (a və b) cəminin iki misli mənimsədilib və alınmış qiymət nəticə olaraq qaytarılıb.

İMTAHAN SUALLARI

1. di dəyişən
2. Dinamik dəyişənlər
3. Ünvan dəyişəni elan etmək
4. Dinamik yaradılma
5. Dinamik silinmə
6. C++ dilində obyektlərin göstəricisi

Ü 1 1 ı üçü ə ö ə ə ə qaydası
ö ə ə ə ə ə lar
Ünvan dəyişənlərin qiymətlərinin dəyişdirilməsi
ın elanı
Funksiyanın mətn kodunun tərtibi
Proqramda hər- hansı funksiyaya müraciət
Funksiya nədir
ın elanı
Funksiyanın təyin olunması
Lokal və Qlobal dəyişənlər
Dəyişənlərin qiymətə görə ötürülməsi.
Dəyişənlərin ünvana görə ötürülməsi
ədir
ölcülü massivin elanı
ölcülü massivin elanı
ölcülü massivin daxil və xaric edilməsi
ölcülü massivin daxil və xaric edilməsi
Massivin təsadüfi ədədlərlə doldurulması
Göstəricilərin massivə istinadı
Göstərici ilə massıv arasında əlaqə
Çox ölçülü massivin yaddaşda yerləşməsi
3 ölçülü massivin elementlərinə müraciət
Simvol və sətirlərin sintaksisi
Sətirlərin elanı və başlanğıc qiymətlərin verilməsi
ılar
struct tipindən olan ünvan dəyişənləri
Strukturlar
ü ə dəyişəni
Siyahının elan edilməsi
Siyahı yaratma qaydası
ılarda dəyişənin NULL qiymətinə mənimsədilməsi
Siyahılardan elementlərin silinməsi.

ə 1 ü 1
ə 1 ü ş 1
ə çı ış
1 ə
1 1 ə ə
ə şə ş ü ə
ə ə ə ə ə ə 1
ə şə ə ə ə ə 1
ə ç ə ə
ö ə ü ü ü ü ə
ə ə
ü üş ü ə ə ə 1
ü ə ə ə 1
ə ə ə
1 ə ə ə 1
Ə ə 1 ü ü ü ə ə ə ə
1

Şə ç
Şə ç
ö ı

ö

62.

ƏDƏBİYYAT:

1. Əhməd Sadıxov ”C++ dilində proqramlaşdırma” Bakı-2012.

2. Allahverdiyev Nailə, Namazov Mənəfəddin ”C dilində proqramlaşdırma”.

3.Mahmud Master, Suha Eriş “C++” İstanbul, 2015-02, 7. Baskı, 150x210, 368 sayfa, Türkçe