O’ZBEKISTON RESPUBLIKASI ALOQA, AXBOROTLASHTIRISH VA TELEKOMUNIKATSIYA DAVLAT QO’MITASI

TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI FARG’ONA FILIALI

”Axborot texnologiyalari” kafedrasi

”C++ da dasturlash”
fanidan

KURS ISHI

Bajardi: 613-12 Guruh talabasi

Uraimjonova Shohida

Qabul qildi: Bazarbayev M.

Hashimov A

Farg’ona-2015

Mavzu: C++ daasturlash tilida dastur yaratish

Reja

1.kirish

a) dasturlash haqida qisqacha ma'lumot.....

2. Asosiy qism.

a) C++ dasturlash tli.....

b) C++ dasturlash tili imkoniyatlari.....

c) C++ dasturlash tilining Massiv imkoniyatlari.....

3. Dasturiy qism

a) Dasturb vazifasi.....

b) Dastur haqida ma'lumot.....

c) Dasturni C++ dasturlash tilida ushbu dasturni yaratish.....

4. Xulosa

5. Foydalanilgan adabiyotlar

Kirish

Tushuntirish izohlariga ega bo’lgan dasturlarni to’g’rilash yoki tuzatish osonroq, chunki ular dastur bilan ishlash uchun qo’shimcha maolumotlarni jamlaydi. Dastur tuzuvchi ko’pincha boshqa tuzuvchining dasturini ko’rib chiqayotgan vaqtda, xatolarni tuzatishda, yoki dastur mantiqni kuzatayotgan vaqtda juda ko’p vaqt sarflaydi. Bu holatda u o’zining meoyorlangan vaqtini juda katta qismni sarflab qo’yadi.

Izohlanmagan dastur - bu taxminimizcha dastur tuzuvchining eng katta xatosidir. Dastur yozish jarayonida izoh berish yaxshi qoidadir. Kelajakda buni hamma dasturchilar hisobga oladilar, degan umiddamiz. Izohning yozishning asosiy qonuni - bu ko’proq izoh berishdir. Albatta, har bir kishi dasturdagi izohni o’qiyotib, shu dastur haqida ko’proq tushunchaga ega bo’ladi. SHuning uchun qancha kerak bo’lsa, shuncha izoh bering.

Izoh berishdan maqsad - dasturni tushunishni osonlashtirishdir.

Ular dasturni kodlashtirganidek yaxshi o’ylagan va ishlab chiqilgan bo’lishi kerak. Izohlar loyihalash va sozlash jarayoni vaqtida va albatta keyinchalik ham kerak bo’ladi. Lekin dastur tugagandan keyin izoh berish foydasiz va maonosizdir. Izoh berishni 3 ta shakli bor:

kiritishda, mundarija va tushuntirishda.

Kiritish izohlari.

har bir dastur, kichik dastur va jarayonlar nima vazifani bajarishligi haqidagi izohlardan boshlanadi. Kiritish izohlaridagi maolumotlar quyidagilardan iborat bo’ladi:

1. Dasturdan maqsad.

2. Dasturni chaqirish va undan foydalanish haqidagi ko’rsatkichlar.

3. Asosiy massiv va o’zgartirgichlarning ro’yxati va vazifasi.

4. Kiritish - chiqarish haqidagi ko’rsatkichlar. Hamma fayllar ro’yxati.

5. Foydalanilayotgan barcha qism dasturlar ro’yxati.

6. Foydalanilgan matematik metodlar va ular haqida maolumotga ega bo’lgan adabiyotlar ro’yxati.

7. Dasturni bajarilgan vaqti haqida maolumot.

8. Kerak bo’lgan xotira hajmi.

9. Operatorga maxsus ko’rsatmalar.

10. Muallif haqida maolumot.

11. Dastur yozilgan sana.

Mundarija.

Agar dastur katta hajmga ega bo’lsa, uning boshiga izoh shaklida mundarija qo’yish maqsadga muvofiqdir. Mundarija o’z ichiga dastur modulining nomini, joyini va funktsiyasini jamlaydi.

Modullar ismiga, izoh Yoki funktsiyalarga ko’rsatmalar bilan taominlangan bo’lishi kerak.

Tushuntirsh izohlari.

Dasturning izohlarsiz tushunib bO’lmaydigan qismlarga tushuntirish izohlari beriladi. Dasturning mantig’ini tushunishdan oldin sonlar yoki shartli operatorlardan ilgari ular haqidagi ko’rsatmalardan iborat izohlar chiqib turishi kerak. Yuqori pog’onada yozilgan 10 ta qatorga 1 ta izohlash qatori o’rtacha meoyor hisoblanadi. Izohlar operatorlar yaratayotgan harakatlarni taoriflamay,

shu operator guruhlari operatorlarini aytib o’tish kerak. Izoh dasturi boshqacha ifodalanmasligi va maolum bir kerakli maolumotlarga ega bo’lishi kerak. Izohlar to’g’ri bo’lib, dstur o’zgarganda o’zgaruvchan bo’lishi kerak. Dasturdagi noto’g’ri izohlar ularni yo’qligiga nasbatan ancha yomonroqdir.

Qatorlarni tashlab ketish - ko’pincha yaxshi bahlanmaydigan, lekin dasturni ko’rinishini yaxshilaydigan metoddir. 1 ta qatorni tashlaganda mantiqan bog’langan operatorlarni har birini ajratamiz. 2 ta qatorni tashlaganda dasturni asosiy mantiqiy qismlarini ajratib olishimiz mumkin bo’ladi. To’ldirmagan qatorlardan foydalanib, dasturdagi alohida bo’laklarni aniq topilishini yengillashtirish.

Oraliq (Probel). Dasturlash tilida oraliqlar ixtiyoriy ravishda qo’yiladi. Oraliqlardan keng foydalanish dasturni o’qishni osonlashtiradi.

Tenglashtirish va ketma - ketlikni belgilash.

Dasturni ketma - ketligini belgilashda biz undagi operatorlarni ketma- ketligi tartiblarida sodir bo’ladigan xatoliklardan forig’ bo’lamiz. Dasturni boshlang’ich qismidan belgilab borishda, bizga uni tekshirishimizda va sozlashimizda katta yordam beradi. CHunki tartib belgi yoki sonlar bizga katta dasturda kerakli qatorlarni aniqlashimizda yordam beradi. Tenglashtirilgan va ketma - ketligi belgilangan dasturlar chiroyli va o’qishga qulay bo’ladi.

O’zgaruvchilarga nom tanlash.

O’zgaruvchilarni nomini shunday tanlash kerakki, bu qismda o’zgaruvchilarni kattaliklari yaqqol bilinib turishi kerak. Dasturni o’qishi oson kechishi uchun ismni to’g’ri tanlashga ham bog’liq bo’ladi. Butun o’zgaruvchilarning nomi I, G, K, L, M, N xarflardan boshlanadi.

O’zgaruvchilar quyidagi turlarga bo’linadi: butun (tselqe), haqiqiy (deystve), kompleksli, belgili (simvolg’nqe).

Fayllar nomi.

Fayllar bilan ishlayotgan vaqtda ularni solishtirish uchun qandaydir perefiks yoki suffiks tanlanadi. Agar har bir fayl o'zining perfiksiga ega bo'lsa, dasturni o'qish oson bo'ladi. Bu perfiks dasturni yozishda mos keluvchi maydonni aniqlab beradi va qaysi ishchi bilan maydonlar mantiqan bog'langanligini aniqlab beradi.

Dasturlarni har xil dasturdagi bir xil fayllarga bir xil nom berilishlari, shu dasturlarni o'zaro solishtirish jarayonlarini osonlashtiradi.

Standart qisqartirishlar.

Dasturni ishlab chiqishda standart qisqartirishlarni kiritish shartdir. CHunki bular begona shaxslarni dasturingizni o'qishini osonlashtiradi. Standart qisqartirishlar dasturchiga kerak bo'lganda eski dasturlarni yangisini yozishga yordamlashadi. Standart qisqartirishlarni ishlab chiqish quyidagilardan iborat: har bir so'zning belgisi xotiraga olinishi kerak. Saqlangan ismdagi so'zlarning umumiy soni 3 dan ortmasligi kerak. Standart qisqartirishlarda har doim so'zlarning bosh xarfi kiritilishi shart. Undosh xarflar unli xarflarga nisbatan axamiyatliroq. So'zning boshi uning oxiridan ko'ra kerakliroq. Standart qisqartirishlar o'z ichiga 6 dan 15 gacha xarflar olishi kerak.

Keyingi satrga ko'chirish (Perenos).

Agar siz yozayotgan qatorga biron bir so'z sig'may qolsa, uni keyingi qatordan boshlang. Albatta ko'chirish ruxsat etiladi. Lekin u dasturni o'qishni va undan foydalanishni qiyinlashtiradi.

Operatorlarni joylashtirish. Bir xilgi dasturlarda bitta qatorga bir necha operatorlarni joylashtirish mumkin. Lekin bu 2 ta sabablarga noqulay: birinchidan, dasturni o'qish qiyinlashadi. Ikkinchidan, u paragraflarga bo'lish

kabi uslubdan foydalanishni almashtiradi. Yana bir sabablardan biri shundaki, sintaksis xato haqida xabar borayotgan vaqtda, qatorning nomeri ham beriladi.

Alfavit bo'yicha ro'yxatni tartiblash.

Dasturlash tilida juda ko'p o'zgaruvchilar qatnashadi va ularni ismi bo'yicha aniqlash tuzuvchining o'ziga bog'liqdir. SHuning uchun tuzuvchining ishini osonlashtirish uchun ro'yxatlar tuziladi va ro'yxatlar 2 ta turga bo'linadi:

1. O'zgaruvchilar turida eolon qilingan o'zgaruvchilar nomi ro'yxati.

2. Jarayon parametrlarining ro'yxati.

Alfavit bo'yicha tartiblashni operator argumentlari ro'yxatida jarayonlarni chaqirib tartiblash ham mumkin. Kichik maolumotlarni tartiblashning boshqa usuli - bu belgilarni nomerlashdir. Ro'yxatlar ustun shaklida bo'lishi kerak. Dasturchi ro'yxatni ustun shaklida tuzish uchun har bir o'zgaruvchilar ro'yxatidagi ismlarning eng ko'p belgilariga mos ravishda ismlar ostida joy qoldirish shart. Bunday joylashtirishlar kirish - chiqishdan tortib har bir berilganlar ro'yxatida bo'lishi kerak.

Qavslar.

Qavslardan to'g'ri foydalanish dasturni o'qishni osonlashtiradi, chunki operatsiyalar bajarilayotgan vaqtda priariatatsiya maolum bo'ladi va bunda dasturchi bir necha qavslardan foydalanishi mumkin, lekin bunda dasturni o'qish va tuzatish qiyinlashadi.

Asosiy qoidalardan biron - gumonli joylarda qavs qo'ying. Bu esa dasturni tushunarli qilib, xato qilishni oldini oladi.

Qator boshidan o'tish.

Operatorlarni yozishda ularni o’rtasidagi bog’liqni ko’rsatish uchun bir satrlar boshidan o’tishlar bajariladi. To’g’ri bajarilgan o’tishlar dasturni tarkibini ko’rsatadi. Bu matnni tushunarli tilda paragrflarga guruhiy so’zlarni mantiqiy bog’lab bo’ladi. TSikllar - o’tishlarda foydalanishdagi oddiy holat. O’tishlarda faqat tsikllarda emas, balki buyruqlarni qulay guruhlashda ham kiritiladi.

Asosiy qism

C++ dasturlash tili C tiliga asoslangan. C esa o'z navbatida B va BCPL tillaridan kelib chiqqan. BCPL 1967 yilda Martin Richards tomonidan tuzilgan va operatsion sistemalarni yozish uchun mo'ljallangan edi. Ken Thompson o'zining B tilida BCPL ning ko'p hossalarini kiritgan va B da UNIX operatsion sistemasining birinchi versiyalarini yozgan.

BCPL ham, B ham tipsiz til bo'lgan. Yan i o'garuvchilarning ma'lum bir tipi bo'lmagan - har bir o'zgaruvchi kompyuter hotirasida faqat bir bayt yer egallagan. O'zgaruvchini qanday sifatda ishlatish esa, yani butun sonmi, kasrli sonmi yoki harfdekmi, dasturchi vazifasi bo'lgan.

C tilini Dennis Ritchie B dan keltirib chiqardi va uni 1972 yili ilk bor Bell Laboratories da, DEC PDP-11 kompyuterida qo'lladi. C o'zidan oldingi B va BCPL tillarining juda ko'p muhim tomonlarini o'z ichiga olish bilan bir qatorda o'zgaruvchilarni tiplashtirdi va bir qator boshqa yangiliklarni kiritdi. Boshlanishda C asosan UNIX sistemalarida keng tarqaldi. Hozirda operatsion sistemalarning asosiy qismi C/C++ da yozilmoqda. C mashina arhitekturasiga bog'langan tildir. Lekin yahshi rejalashtirish orqali dasturlarni turli kompyuter platformalarida ishlaydigan qilsa bo'ladi.

1983 yilda, C tili keng tarqalganligi sababli, uni standartlash harakati boshlandi. Buning uchun Amerika Milliy Standartlar Komiteti (ANSI) qoshida X3J11 tehnik komitet tuzildi. Va 1989 yilda ushbu standart qabul qilindi. Standartni dunyo bo'yicha keng tarqatish maqsadida 1990 yilda ANSI va Dunyo Standartlar Tashkiloti (ISO) hamkorlikda C ning ANSI/ISO 9899:1990 standartini qabul qilishdi.

Shu sababli C da yozilgan dasturlar kam miqdordagi o'zgarishlar yoki umuman o'zgarishlarsiz juda ko'p kompyuter platformalarida ishlaydi.

C++ 1980 yillar boshida Bjarne Stroustrup tomonidan C ga asoslangan tarzda tuzildi. C++ juda ko'p qo'shimchalarni o'z ichiga olgan, lekin eng asosiysi u ob'ektlar bilan dasturlashga imkon beradi.

Dasturlarni tez va sifatli yozish hozirgi kunda katta ahamiyat kasb etmoda. Buni ta'minlash uchun ob'ektli dasturlash g'oyasi ilgari surildi. Huddi 70-chi yillar boshida strukturali dasturlash kabi, programmalarni hayotdagi jismlarni modellashtiruvchi ob'ektlat orqali tuzish dasturlash sohasida inqilob qildi.

C++ dan tashqari boshqa ko'p ob'ektli dasturlshga yo'naltirilgan tillar paydo bo'ldi. Shulardan eng ko'zga tashlanadigani Xerox ning Palo Altoda joylashgan ilmiy-qidiruv markazida (PARC) tuzilgan Smalltalk dasturlash tilidir. Smalltalk da hamma narsa ob'ektlarga asoslangan. C++ esa gibrid tildir. Unda C ga o'hshab strukturali dasturlash yoki yangicha, ob'ektlar bilan dasturlash mumkin. Yangicha deyishimiz ham nisbiydir. Ob'ektli dasturlash falsafasi paydo bo'lganiga ham yigirma yildan oshayapti.

C++ funksiya va ob'ektlarning juda boy kutubhonasiga ega. Yani C++ da dasturlashni o'rganish ikki qismga bo'linadi. Birinchisi bu C++ ni o'zini o'rganish, ikkinchisi esa C++ ning standart kutubhonasidagi tayyor ob'ekt/funksiyalarni qo'llashni o'rganishdir

.

C++ dasturlash tili imkoniyatlar

C++ sistemasi asosan quyidagi qismlardan iborat. Bular dasturni yozish redaktori, C++ tili va standart kutubhonalardir. C++ dasturi ma'lum bir fazalardan o'tadi. Birinchisi dasturni yozish va tahrirlash, ikkinchisi preprosessor amallarini bajarish, kompilyatsiya, kutubhonalardagi ob'ekt va funksiyalarni dastur bilan bog'lash (link), hotiraga yuklash (load) va bajarish (execute).

C++ da arifmetik amallar

Ko'p programmalar ijro davomida arifmetik amallarni bajaradi. C++ dagi amallar

quyidagi jadvalda berilgan. Ular ikkita operand bilan ishlatdi.

C++ dagi amal Arifmetik operator Algebraik ifoda C++ dagi ifodasi

Qo'shish + h+19 h+19

Ayirish - f-u f-u

Ko'paytirish * sl s*l

Bo'lish / v/d, vod v/d

Modul olish % k mod 4 k%4
Bularning ba'zi birlarinig hususiyatlarini ko'rib chiqaylik. Butun sonli bo'lishda, yani bo'luvchi ham, bo'linuvchi ham butun son bo'lganda, javob butun son bo'ladi. Javob yahlitlanmaydi, kasr qismi tashlanib yuborilib, butun qismining o'zi qoladi.
Modul operatori (%) butun songa bo'lishdan kelib chiqadigan qoldiqni beradi. x%y ifodasi x ni y ga bo'lgandan keyin chiqadigan qoldiqni beradi. Demak, 7%4 bizga 3 javobini beradi. % operatori faqat butun sonlar bilan ishlaydi. Vergulli (real) sonlar bilan ishlash uchun "math.h" kutubhonasidagi fmod funksiyasini qollash kerak.
C++ da qavslarning ma'nisi huddi algebradagidekdir. Undan tashqari boshqa boshqa algebraik ifodalarning ketma-ketligi ham odatdagidek. Oldin ko'paytirish, bo'lish va modul olish operatorlari ijro ko'radi. Agar bir necha operator ketma-ket kelsa, ular chapdan o'nga qarab ishlanadi. Bu operatorlardan keyin esa qo'shish va ayirish ijro etiladi.
Misol keltiraylik. k = m * 5 + 7 % n / (9 + x);
Birinchi bo'lib m * 5 hisoblanadi. Keyin 7 % n topiladi va qoldiq (9 + x) ga bo'linadi. Chiqqan javob esa m * 5 ning javobiga qo'shiladi. Qisqasini aytsak, amallar matematikadagi kabi. Lekin biz o'qishni osonlashtirish uchun va hato qilish ehtimolini kamaytirish maqsadida qavslarni kengroq ishlatishimiz mumkin. Yuqoridagi misolimiz quyidagi ko'rinishga ega bo'ladi.
k = (m * 5) + ((7 % n) / (9 + x));

Mantiqiy solishtirish operatorlari

C++ bir necha solishtirish operatorlariga ega.
Algebraik ifoda C++ dagi operator C++ dagi ifoda Algebraik ma'nosi
tenglik guruhi
= == x==y x tengdir y ga
teng emas != x!=y x teng emas y ga
solishtirish guruhi
> > x>y x katta y dan
< < x<y x kichkina y dan
katta-teng >= x>=y x katta yoki teng y ga
kichik-teng <= x<=y x kichik yoki teng y ga
==, !=, >= va <= operatorlarni yozganda oraga bo'sh joy qo'yib ketish

sintaksis hatodir. Yani kompilyator dasturdagi hatoni ko'rsatib beradi va uni tuzatilishini talab qiladi. Ushbu ikki belgili operatorlarning belgilarining joyini almashtirish, masalan <= ni =< qilib yozish ko'p hollarda sintaksis hatolarga olib keladi. Gohida esa != ni =! deb yozganda sintaksis hato vujudga ham, bu mantiqiy hato bo'ladi. Mantiqiy hatolarni kompilyator topa olmaydi. Lekin ular programma ishlash mantig'ini o'zgartirib yuboradi. Bu kabi hatolarni topish esa ancha mashaqqatli ishdir (! operatori mantiqiy inkordir). Yana boshqa hatolardan biri tenglik operatori (==) va tenglashtirish, qiymat berish operatorlarini (=) bir-biri bilan almashtirib qo'yishdir. Bu ham juda ayanchli oqibatlarga olib keladi, chunki ushbu hato aksariyat hollarda mantiq hatolariga olib keladi

Yangi stildagi e'lon fayllari va ismlar sohasi tushunchasi

C++ ning standarti .h bilan tugaydigan (stdio.h ...) standart kutubhona e'lon fayllarini yangittan nomlab chiqdi. Bunda .h qo'shimchasi olib tashlandi.
C dan qolgan fayllar ismiga esa c harfi qo'shildi.
Misol uchun:
iostream.h -> iostream
string.h -> cstring
stdlib.h -> cstdlib
time.h -> ctime
C dan meros qolgan kutubhona 18 ta e'lon fayli orqali berilgan. C++ ga tegishli standart kutubhonada esa 32 ta e'lon fayl bor.
Fayllarni yangittan belgilashdan maqsad kutubhodadagi funksiya va ob'ektlarni std deb ataluvchi ismlar sohasiga (namespace) kiritishdir.
Ismlar sohasining o'zi ham nisbatan yangi tushuncha. Ismlar sohasini alohida dastur qismlari deb faraz qilsak boladi. Boshqa-boshqa sohalarda ayni ismli funksiya, o'zgaruvchi nomlari va ob'ektlar berilishi mumkin. Va bunda hech qanday ismlar to'qnashuvi sodir bo'lmaydi. Misol uchun bizda global, std va fun::obj degan ism sohalari bo'lsin. Ularning har birining ichida esa cout nomli ob'ekt aniqlangan bo'lsin. C++ da to'liq aniqlangan ism (fully qualified name) degan tushuncha bor. Shunga ko'ra har bir cout ob'ektinig to'liq ismi quyidagicha bo'ladi:
Ismlar sohasi ob'ekt
global ::cout

std std::cout
fun::obj fun::obj::cout
:: operatori sohalarni bog'lash uchun qo'llaniladi. fun::obj nomli ismlar sohasida obj fun ichida joylashgan ism sohasidir. Global ismlar sohasida aniqlangan funksiya va boshqa turdagi dastur birliklariga programmaning istalgan yeridan yetishsa bo'ladi. Masalan global ismlar sohasida e'lon qilingan int tipidagi k ismli o'zgaruvchimiz bol'sa, uning ustidan dasturning hohlagan blokida amal bajarsak bo'ladi.
Ismlar sohasi mehanizmi dasturchilarga yangi kutubhonalarni yozish ishini ancha osonlashtiradi. Chunki yangi kutubhonada ayni ismlar qo'llanishiga qaramay, ismlar konflikti yuz bermaydi. Dastur yoki kutubhona yozganda yangi ismlar sohasini belgilash uchun
namespace istalgan_ism {
...
foo();
int k;
String str;
...
}
deb yozamiz. Dasturimizda ushbu ismlar sohasida aniqlangan o'zgaruvchilarni ishlatish uchun ularning to'liq ismini yozishimiz kerak. Masalan:
istalgan_ism::foo();
Ammo bu usul ancha mashaqqatli bo'ladi. Har bir funksiya yoki o'zgaruvchi oldiga uning to'liq aniqlangan ismini yozish ko'p vaqt oladi. Buning o'rniga biz
using namespace istalgan_ism;
deb yozib o'tsak kifoya. using (ishlatish, qo'llash) buyrug'i bizning ismlar sohamizni dasturimiz ichiga tanishtiradi. Eng asosiysi biz bu amalni sohada aniqlangan va biz qo'llamoqchi bo'lgan ismlarning ilk chaqirig'idan oldin yozishimiz kerak. C++ ning standart kutubhonasida aniqlangan ifodalarni qo'llash uchun biz
using namespace std;
deymiz. Va albatta qo'llanilayotgan e'lon fayllari yangi tipda bo'lishi kerak. Endi bu tushunchalarni ishlatadigan bir dasturni keltiraylik.
//Yangi tipdagi e'lon fayllari va ismlar sohasini qo'llash.
include <iostream>

```
using namespace std;
int main()
{
std::cout << "Hello!\n";
cout << "Qale!";
return (0);
}
```
Ekranda:
Hello!
Qale!
std::cout << "Hello\n"; satrida biz chiqish oqimi ob'ekti cout ning to'liq ismini qo'lladik. Keyingi satrda esa biz yana ayni ob'ektni ishlatdik, lekin endi uni to'liq atab o'tirmadik, chunki biz std ismlar sohasini using bilan e'lon qilib bo'ldik.
Ismlarni global ismlar sohasida e'lon qilish uchun ularni blok va funkisiyalar tashqarisida aniqlash kerak. Masalan:
```
# include <stdio.h>
int i;
int main()
{
...
int k;
...
return (0);
}
```
Bu yerda i global ismlar sohasida joylashgan, k esa main() funksiyasiga tegishli. O'zgaruvchilarni global ism sohasida aniqlashning boshqa yo'li, ularni nomsiz ismlar sohasida belgilashdir. Yani:
```
namespace {
int j;
}
```
j o'zgaruvchisi global boldi. Uni ishlatish uchun:
```
::j = ::j + 7;
```
:: operatorini qo'llashimiz mumkin, yoki oddiygina qalib faqat o'zini:
```
j = j * 9;
```
kabi yozishimiz mumkin. Ammo agar biz ishlayatgan dastur blokida ayni ismli

o'zgaruvchi bo'lsa, masalan j, unda ushbu lokal aniqlangan j bizning global j imizni berkitib qo'yadi. Biz j ni o'zini qo'llasak, lokal j ga murojat qilgan qilgan bo'lamiz. Global j ni ishlatish uchun endi :: operatorini qo'llashga majburmiz. Bu mulohazalar umuman olganda boshqa ismlar sohalarini
ishlatganimizda ham o'rinlidir.

Boshqaruv ifodalari

Bu bo'limda biz strukturali dasturlashning asosiy prinsip va qismlarini ko'rib chiqamiz. Ma'lum bir dasturni yozish uchun belgilangan qadamlarni bosib o'tish kerak. Masala aniqlangandan so'ng uni yechish uchun mo'ljallangan algoritm tuziladi. Keyin esa psevdokod yoziladi. Psevdokod algoritmda bajariladigan qadamlarni ko'rsatadi. Bunda faqat bajariladigan ifodalar ko'rib chiqiladi. Psevdokodda o'zgaruvchi e'lonlari yoki boshqa ma'lum bir dasturlash tiliga mansub bo'lgan yordamchi amallar bo'lmaydi.
Psevdokodni yozish dasturlashni ancha osonlashtiradi, algoritm mantig'ini tushunishga va uni rivojlanritishga katta yordam beradi. Misol uchun bir dasturning rejasi va psevdokodi 3-4 oy yozilgan bo'lsa va yuqori darajada detallashtirilgan bo'lsa, ushbu dasturning C++ yoki boshqa tildagi kodini yozish 2-3 hafta vaqt oladi halos. Bu yozilgan programmada hato ancha kam bo'ladi, uni keyinchalik takomillashtirish arzonga tushadi. Hozirgi paytda dastur o'zgarishi fafqulotda hodisa emas, balki zamon talabidir.

Dastur ijro strukturalari

Asosan dasturdagi ifodalar ketma-ket, navbatiga ko'ra ijro etiladi. Gohida bir shart bajarilishiga ko'ra, ijro boshqa bir ifodaga o'tadi. Navbatdagi emas, dasturning boshqa yerida joylashgan ifoda bajariladi. Yani sakrash yoki ijro ko'chishi vujudga keladi.
60-chi yillarga kelib, dasturlardagi ko'pchilik hatolar aynan shu ijro ko'chishlarining rejasiz ishlatilishidan kelib chiqishi ma'lum bo'ldi. Bunda eng katta aybdor deb bu ko'shishlarni amalga oshiruvchi goto (...ga bor) ifodasi belgilandi. goto dastur ijrosini deyarli istalgan yerga ko'chirib yuborishi mumkin. Bu esa programmani o'qishni va uning strukturasini murakkablashtirib yuboradi. Shu sababli "strukturali dasturlash" atamasi

"goto ni yo'q qilish" bilan tenglashtirilardi. Shuni aytib o'tish kerakki, goto kabi shartsiz sakrash amallarini bajaruvchi ifodalar boshqa dasturlash tillarida ham bor.

Tadqiqotlar shuni ko'rsatdiki, istalgan programma goto siz yozilishi mumkin ekan. goto siz yozish uslubi strukturali dasturlash deb nom oldi. Va bunday dastur yozish metodi katta iqtisodiy samara beradi.

Strukturali dasturlash asosi shundan iboratki, har bir programma faqatgina uch hil boshqaruv strukturalaridan iboratdir.Bular ifodalarni ketma-ket ijro etish strukturasi (sequence structure), tanlash strukturasi (selection structure) va amalni qayta ijro etish strukturasidir (repetition structure).

Ifodalarni ketma-ket ijro etish strukturasi C++ tomonidan ta'minlanadi. Normal sharoitda C++ ifodalari dasturdagi navbatiga ko'ra bajariladi.

Tanlash buyruqlari uchtadir. Bular if, if/else va switch dir. Qayta ijro etish buyruqlari gurugiga ham uchta a'zo bor, bular while, do/while va for. Bularni har birini keyinroq tahlil qilib chiqamiz.

Yuqoridagi buyruqlar nomlari C++ dasturlash tilining mahsus so'zlaridir. Dasturchi bu so'zlarni o'zgaruvchi yoki funksiyalar nomi sifatida qo'llashi ta'qiqlanadi. Quyida C++ ning ajratilgan so'zlarining to'liq ro'yhati berilgan.

C++ va C ga tegishli:

auto do goto signed unsigned
break double if sizeof void
case else int static volatile
char enum long struct while
const extern register switch
continue float return typedef
default for short union

Faqat C++ ga qarashli:

asm explicit operator this virtual
bool false private throw wchar_t
catch friend protected true
class inline public try
const_cast mutable reinterpret_cast typeid
delete namespace static_cast typename
dynamic_cast new template using

C++ dagi yetita boshqaruv strukturasini aytib o'tdik. Ular bittagina

boshlanish nuqtasiga va bittagina chiqish nuqtasiga egadirlar. Demak biz bu dastur bo'laklarini ketma-ket ulab ketishimiz mumkin. Boshqaruv strukturalarining bu kabi ulanishini devorning g'ishtlarini ustma-ust qalashga ham taqqoslasak bo'ladi. Yoki biz bu bloklarni bir-birining ichiga joylashtirishimiz mumkin. Bu kabi qo'llashish ikkinchi uslub bo'ladi. Mana shu ikki yo'l bilan bog'langan yetita blok yordamida biz istalgan dasturimizni yoza olamiz

Massivlar

Bu qismda dasturdagi ma'lumot strukturalari bilan tanishishni boshlaymiz. Dasturda ikki asosiy tur ma'lumot strukturalari mavjuddir. Birinchisi statik, ikkinchisi dinamikdir. Statik deganimizda hotirada egallagan joyi o'zgarmas, dastur boshida beriladigan strukturalarni nazarda tutamiz. Dinamik ma'lumot tiplari dastur davomida o'z hajmini, egallagan hotirasini o'zgartirishi mumkin.
Agar struktura bir hil kattalikdagi tiplardan tuzilgan bo'lsa, uning nomi massiv (array) deyiladi. Massivlar dasturlashda eng ko'p qo'laniladigan ma'lumot tiplaridir. Bundan tashqari strukturalar bir necha farqli tipdagi o'zgaruvchilardan tashkil topgan bo'lishi mumkin. Buni biz klas (Pascalda record) deymiz. Masalan bunday strukturamiz ichida odam ismi va yoshi bo'lishi mumkin.
Bu bo'limda biz massivlar bilan yaqindan tanishib o'tamiz. Bu bo'limdagi massivlarimiz C uslubidagi, pointerlarga (ko'rsatkichlarga) asoslan strukturalardir. Massivlarning boshqa ko'rinishlarini keyingi qismlarda o'tamiz.
Massivlar hotirada ketma-ket joylashgan, bir tipdagi o'zgaruvchilar guruhidir. Alohida bir o'zgaruvchini ko'rsatish uchun massiv nomi va kerakli o'zgaruvchi indeksini yozamiz. C/C++ dagi massivlardagi elementlar indeksi har doim noldan boshlanadi. Bizda char tipidagi m nomli massiv bor bo'lsin. Va uning 4 dona elementi mavjud bo'lsin. Shemada bunday ko'rsataylik:
m[0] -> 4
m[1] -> -44
m[2] -> 109
m[3] -> 23
Ko'rib turganimizdek, elementga murojaat qilish uchun massiv nomi va []

qavslar ichida element indeksi yoziladi. Bu yerda birinchi element qiymati 4, ikkinchi element - 1 nomerli indeksda -44 qiymatlari bor ekan. Ohirgi element indeksi n-1 bo'ladi (n - massiv elementlari soni).
[] qavslar ichidagi indeks butun son yoki butun songa olib keluvchi ifoda bo'lmog'i lozim. Masalan:

```
...
int k = 4, l = 2;
m[ k-l ] = 77; // m[2] = 77
m[3] *= 2; // m[3] = 46
double d = m[0] * 6; // d = 24
cout << m[1]; // Ekranda: -44
...
```

Massivlarni ishlatish uchun ularni e'lon qilish va kerak bo'lsa massiv elementlarini initsalizatsiya qilish kerak. Massiv e'lon qilinganda kompilyator elementlar soniga teng hajmda hotira ajratadi. Masalan yuqorida qo'llanilgan char tipidagi m massivini e'lon qilaylik.

```
char m[4];
```

Bu yerdagi 4 soni massivdagi elementlar miqdorini bildiradi. Bir necha massivni e'londa bersak ham bo'ladi:

```
int m1[4], m2[99], k, l = 0;
```

Massiv elementlari dastur davomida initsalizatsiya qilishimiz mumkin, yoki boshlang'ich qiymatlarni e'lon vaqtida, {} qavslar ichida ham bersak bo'ladi. {} qavslardagagi qiymatlar massiv initsalizaytsiya ro'yhati deyiladi.

```
int n[5] = {3, 5, -33, 5, 90};
```

Yuqorida birinchi elementning qiymati 3, ikkinchiniki 5 ... ohirgi beshinchi element qiymati esa 90 bo'ldi. Boshqa misol:

```
double array[10] = {0.0, 0.4, 3.55};
```

Bu yerdagi massiv tipi double bo'ldi. Ushbu massiv 10 ta elementdan iboratdir. {} qavslar ichida esa faqat boshlangich uchta element qiymatlari berildi. Bunday holda, qolgan elementlar avtomatik tarzda nolga tenglashtiriladi. Bu yerda aytib o'tishimiz kerakki, {} qavslar ichida berilgan boshlangish qiymatlar soni massivdagi elementlar sonidan katta bo'lsa, sintaksis hatosi vujudga keladi. Masalan:

```
char k[3] = {3, 4, 6, -66, 34, 90}; // Hato!
```

Uch elementdan iborat massivga 6 dona boshlangich qiymat berilyapti, bu hatodir. Boshqa misolni ko'rib chiqaylik:

int w[] = {3, 7, 90, 78};
w nomli massiv e'lon qilindi, lekin [] qavslar ichida massivdagi elementlar soni berilmadi. Bunday holda necha elementga joy ajratishni kompilyator {} qavslar ichidagi boshlangich qiymatlar miqdoriga qarab biladi. Demak, yuqoridagi misolda w massivimiz 4 dona elementdan iborat bo'ladi.
E'lon davridagi massiv initsalizatsiya ro'yhati dastur ijrosi vaqtidagi initsalizatsiyadan ko'ra tezroq ishlaydigan mashina kodini vujudga keltiradi.
Bir misol keltiraylik.

```
// Massivlar bilan ishlash.
# include <iostream.h>
# include <iomanip.h>
const int massiv = 8; // massiv kattaligi uchun konstanta
int k[massiv];
char c[massiv] = {5,7,8,9,3,44,-33,0}; // massiv initsalizatsiya ro'yhati
int main()
{
for (int i = 0; i < massiv; i++) {
k[i] = i + 1; // dastur ichida inisalizatsiya
}
for (int j = 0; j < massiv; j++) {
cout << k[j]
<< setw(4)
<< c[j]
<< endl;
}
return (0);
}
```

Ekranda:

```
1 5
2 7
3 8
4 9
5 3
6 44
7 -33
8 0
```

Yuqorida <iomanip.h> faylini dasturimizga kiritdik. Bu e'lon faylida standart kirish/chiqish oqimlari bilan ishlaydigan buyruqlar berilgan. Dasturimizda qo'llanilgan setw() manipulyatori chiqish oqimiga berilayatgan ma'lumotlarning eng kichik kengligini belgilaydi, biz setw() parametrini 4 deb berdik, demak c[] massivi elementlari 4 harf kenglikda ekranga bosiladilar. Agar kenglik kamlik qilsa, u kattalashtiriladi, agar bo'sh joy qolsa, elementlar chapga yondashilib yoziladi. Biz <iomanip.h> va manipulyatorlarni keyinroq to'la ko'rib chiqamiz.

Misolimizda massiv nomli konstantani qo'lladik. Uning yordamida massiv chegaralarini va for strukturasidagi chegaraviy qiymatlarni berdik. Bunday o'zgarmasni qo'llash dasturda hatoga yo'l qo'yishni kamaytiradi. Massiv chegarasi o'zgarganda, dasturning faqat bir joyiga o'zgarish kiritiladi.

Massiv hajmi e'lonida faqat const sifatli o'zgaruvchilar qo'llanilishi mumkin.

Massivlar bilan ishlaganda eng ko'p yo'l qoyiladigan hato bu massivga 0 dan kichkina va (n-1) dan (n: massivdagi elementlar soni) katta indeks bilan murojaat qilishdir. Bunday hato dastur mantig'i hatosiga olib keladi. Kompilyator bu turdagi hatolarni tekshirmaydi. Keyinroq o'zimiza yozgan massiv klaslarida ushbu hatoni tekshiriladigan qilishimiz mumkin.

10 ta sonning tushish ehtimilini ko'rsaturvchi dastur yozaylik.

```
// Ehtimollar va massivlar
# include <iomanip.h>
# include <iostream.h>
# include <time.h>
# include <stdlib.h>
int main ()
{
const int massivHajmi = 10;
int m[massivHajmi] = {0}; // hamma 10 ta element
// 0 ga tenglashtirildi
srand( time(NULL) );
for(int i = 0; i < 1000; i++) {
++m[ rand() % 10 ];
}
for(int j = 0; j < massivHajmi; j++) {
cout << j << setw(4) << m[j] << endl;
}
```

```
return (0);
}
```
Ekranda:
```
0 96
1 89
2 111
3 97
4 107
5 91
6 100
7 118
8 99
9 92
```
Ko'rib turganimizdek, sonlarning tushish ehtimoli nisbatan tengdir. Albatta, bu qiymatlar dasturning har yangi ishlashida o'zgaradi.
```
++m[ rand() % 10 ];
```
Yozuvi bilan biz massivning rand() % 10 indeksli elementini birga oshirmoqdamiz. Bunda rand () % 10 ifodasidan chiqadigan qiymatlar [0;9] ichida
yotadi.
Satrlar, yani harflar ketma-ketligi ("Toshkent", "Yangi yilingiz bilan!"...) C/C++ da char tipidagi massivlar yordamida beriladi. Bunday satrlar bilan islovlar juda tez bajariladi. Chunki ortiqcha tekshirishlar bajarilmaydi. Bundan tashqari C++ da ancha rivojlangan String klasi mavjuddir, u oddiy char bilan berilgan satrlardan ko'ra qulayroqdir. Lekin ushbu klas ko'proq joy egallaydi va massivli satrlardan ko'ra sekinroq ishlaydi. String klasini keyingi qismlarda o'tamiz. Qolaversa, satrlar bilan ishlash uchun biz o'zimiz klas yoki struktura yozishimiz mumkin. C dan meros bo'lib qolgan satrlar ustida amallar bajarish uchun biz dasturimizga <string.h> (yangi ismi <cstring>) e'lon faylini kiritishimiz kerak. Ushbu e'lon faylida berilgan funksiyalar bilan keyingi bo'limda ishlaymiz.
Harflar, yani literalar, aytib o'tganimizdek, C++ da char tipi orqali beriladi. Literalar apostroflarga ('S', '*' ...) olinadi. Satrlar esa qo'shtirnoqlarga olinadi. Satrlar e'loniga misol beraylik.
```
char string[] = "Malibu";
char *p = "Ikkinchi!?!";
```

Satrlarni yuqoridagi ikkita yo'l bilan initsalizatsiya qilsa bo'ladi.
Satrlar ikkinchi uslubda e'lon qilinganda, yani pointer mehanizmi qo'llanilganda, ba'zi bir kompilyatorlar satrlarni hotiraning konstantalar saqlanadigan qismiga qo'yadi. Yani ushbu satrlarga o'zgartirish kiritilishi ta'qiqlanadi. Shu sababli satrlarni hardoim o'zgartirish imkoni bo'lishi uchun ularni char tipidagi massivlar ko'rinishida e'lon qilish afzaldir.
Satrlar massiv yordamida berilganda, ularning uzunligi noma'lumdir.
Shu sababli satrning tugaganligini bildirish uchun satr ohiriga mahsus belgi nol literasi qo'yiladi. Uning dastursa belgilanishi '\0' ko'rinishga ega.
Demak, yuqorida berilgan "Malibu" satriga ohiriga '\0' belgisi qo'shiladi, yani massivning umumiy uzunligi "Malibu":6 + '\0':1 = 7 ta char tipidagi elementga teng bo'ladi. Satrlarni massiv initsalizatsiya ro'yhati ko'rinishida ham bersak bo'ladi:

```
char c[6] = {'A', 'B', 'C', 'D', 'E' , '\0'};
...
cout << c;
...
```

Ekranda:

```
ABCDE
```

Biz cout bilan c ning qiymati ekranga bosib chiqardik. Aynan shunday klaviaturadan ham o'qib olishimiz mumkin:

```
char string[80];
cin >> string;
```

Eng muhimi satr bilan '\0' belgisi uchun yetarli joy bo'lishi kerak.
Satrlar massiv yordamida berilganligi uchun, alohida elementlarga indeks orqali yetishish mumkin, masalan:

```
char k[] = "Bahor keldi, gullar ochildi.";
for (int i = 0; k[i] != '\0'; i++)
if ( (i mod 2) == 0 ) // juft sonlar uchun haqiqat bo'ladi
cout << k[i] << " ";
```

Ekranda:

```
B h r k l i u l r o h l i
```

Yuqoridagi misolda, sikl tugashi uchun k[i] element '\0' belgiga teng bo'lishi kerak.

Funksiyalarning massiv kirish parametrlari

Funksiyalarga massivlarni kirish argument sifatida berish uchun parametr e'lonida [] qavslar qo'yiladi. Masalan:

```
...
void sortArray(int [], int ); // funksiya e'loni
void sortArray(int n[], int hajm) { // funksiya aniqlanishi
...
}
...
```

Dasturda esa, funksiya chaqirilganda, massivning faqat ishmi beriladi halos, [] qavslarning keragi yo'q.

```
int size = 10;
int array[size] = {0};
...
void sortArray(array, size); // funksiya chaqirig'i,
// faqat massiv ismi - array berildi
...
```

Funksiyaga massivlarni berganimizda, eng katta muammo bu qanday qilib massivdagi elementlari sonini berishdir. Eng yaxshi usul bu massiv kattaligini qo'shimcha kirish parametri orqali funksiyaga bildirishdir. Bundan tashqari, massiv hajmini global konstanta orqali e'lon qilishimiz mumkin. Lekin bu ma'lumotni ochib tashlaydi, global sohani ortiqcha narsalar bilan to'ldirib tashlaydi. Undan tashqari massiv hajmini funksiyaning o'ziga yozib qoyishimiz mumkin. Biroq bunda bizning funksiyamiz faqat bitta kattalikdagi massivlar bilan ishlaydigan bo'lib qoladi. Yani dasturimiz dimamizmni yo'qotadi.
Klaslar yordamida tuzilgan massivlar o'z hajmini biladi. Agar bunday ob'ektlarni qo'llasak, boshqa qo'shimcha parametrlarni qo'llashimizning keragi yo'q.
Funksiyalarga massivlar ko'rsatkich ko'rinishida beriladi. Buni C++, biz ko'rsatmagan bo'lsak ham, avtomatik ravishda bajaradi. Agar massivlar qiymat bo'yicha chaqirilganda edi, har bir massiv elementining nushasi olinishi kerak bo'lardi, bu esa dastur ishlash tezligiga salbiy ta'sir ko'rsatar edi.
Lekin massivning alohida elementi argument o'rnida funksiyaga berilganda, ushbu element, aksi ko'rsatilmagan bo'lsa, qiymat bo'yicha beriladi. Masalan:

```
...
double m[3] = {3.0, 6.88, 4.7};
void foo(double d){
```

```
...
}
...
int main()
{
...
void foo(m[2]); // m massivining uchinchi elementining qiymati - 4.7 berildi
...
return (0);
}
```

Agar kiritilayatgan massiv funksiya ichida o'zgarishi ta'qiqlansa, biz funksiya massiv parametri oldiga const sifatini qo'ysak bo'ladi:

```
foo(const char []);
```

Bunda funksiyaga kiradigan massiv funksiya tomonidan o'zgartirilmaydi. Agar o'zgartirishga urinishlar bo'lsa, kompilyator hato beradi.

Daturiy qism

Dastur vazifasi:

Biz tuzmoqchi bo’lgan datur quyidagi vazifani bajarish uchun xizmat qiladi.

Aylana bo’yicha 1 ta son yozilgan: $a_1,a_2,….,a_{12}$,. Agar k-tartib raqamidan boshlab, ularni qo’shib yozsak, X_k vector xosil bo’ladi:

$X_k = (a_k,a_{k+1}, a_{k+11})$.

Bu yerda a_{13},a_{14} ni a_2 va x.k. bildiradi. Agar birinchi tengmas juftdayoq $a_{k+1}<a_{p+I}$ (i=0.1,…) bo’lsa X_k vector X_p vectoedan kichik deb xisoblanadi.

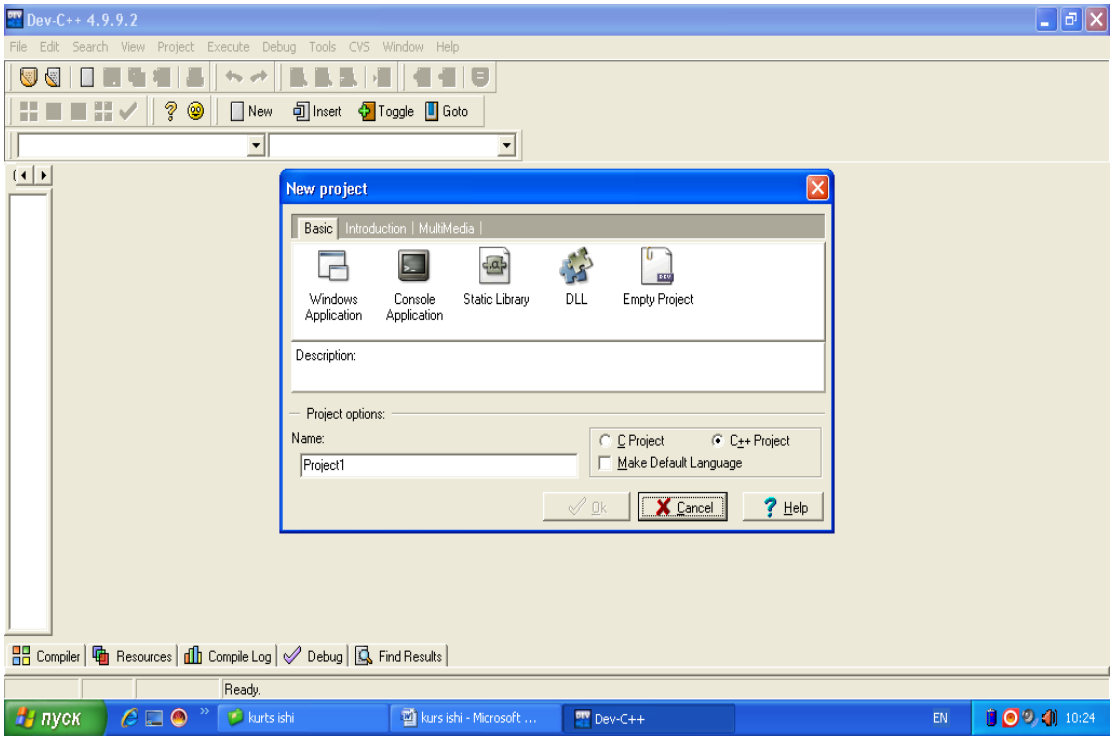
Shunday <<k>> ni toppish kerakki, vector X_k eng kichik bo’lsin.

Dastur haqida ma’lumot

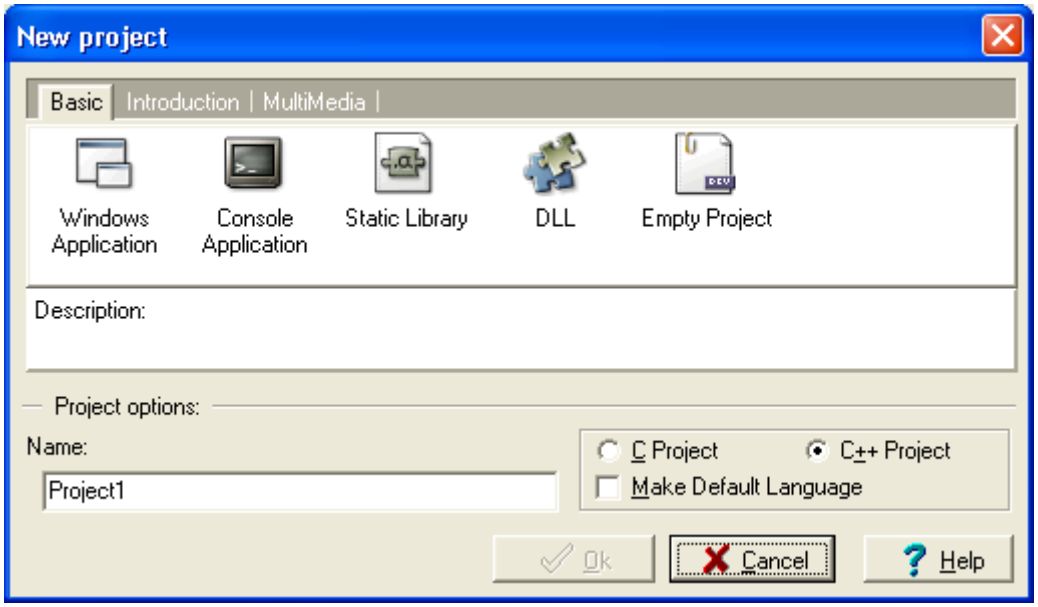
Dastureni C++ dasturlash tilida tuzish uchun quyidagi buyruqlarni bajaramiz.

1. Dasturda ishlashimiz uchun bizga dasturning asosiy ya’ni tana qismini tuzish uchun bizga asosiy ishchi oyna kerak biz uni menyular

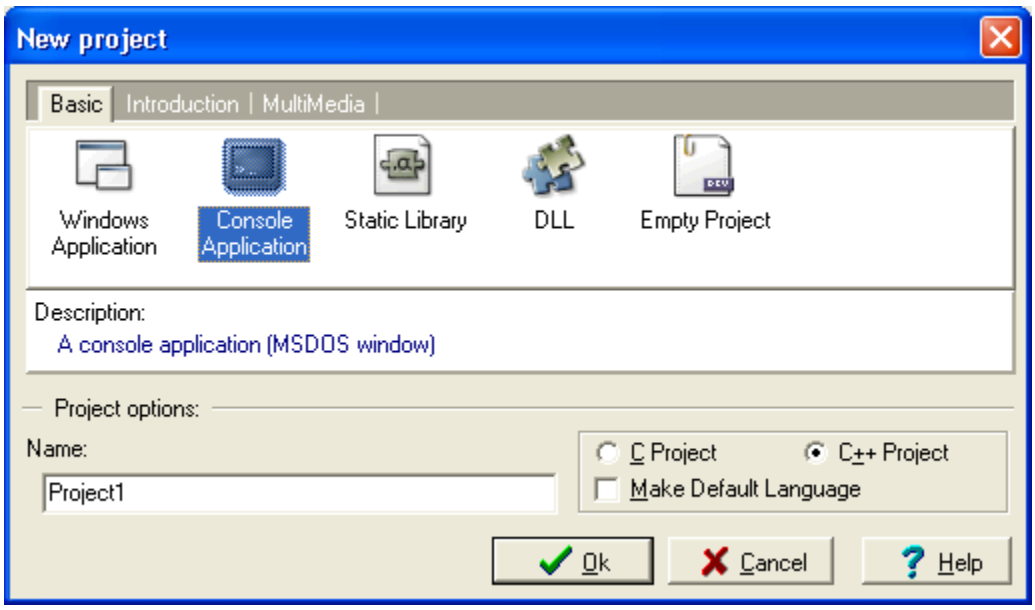
qatorining birinchi menyusi bo’lgan <File> mrnyusida sichqonchani chap tugmasini bir marta bosamiz va <New> undan <Project> ga sichqonchani chap tugmasini bir marta bosamiz quyidagi oynaga ega bolamiz.



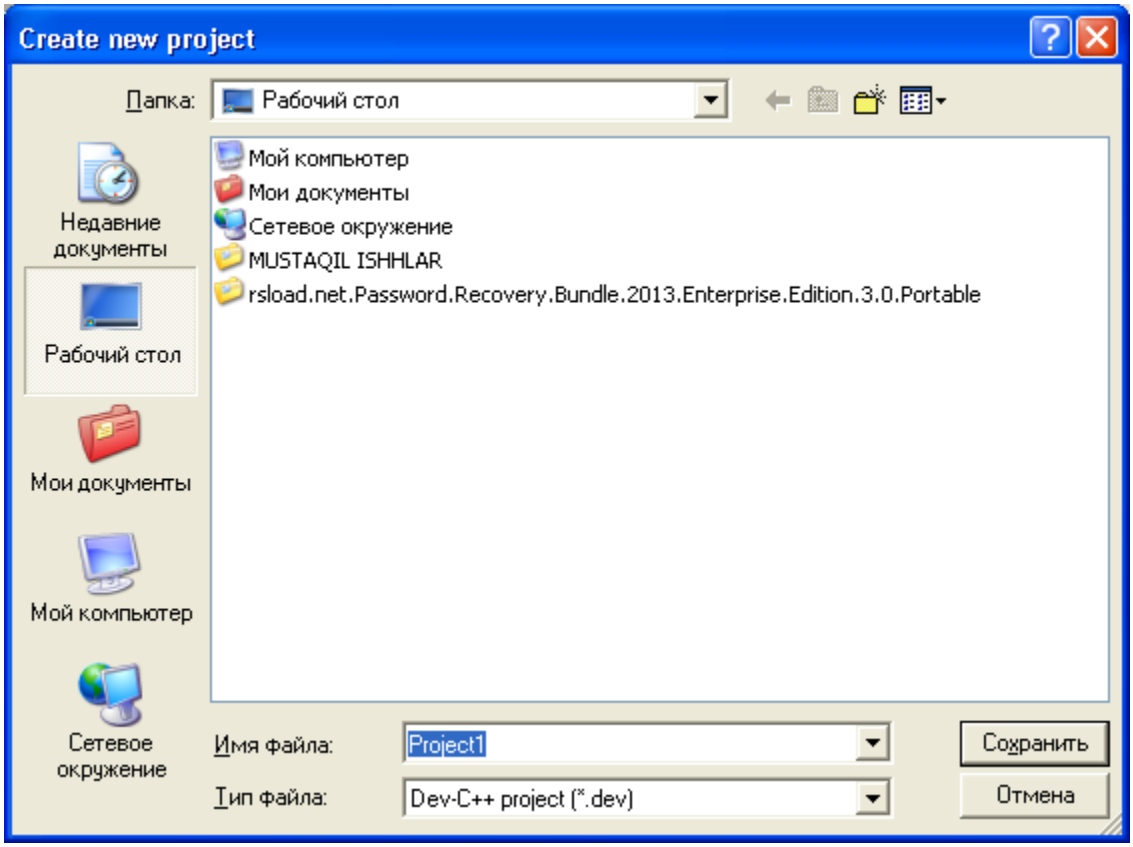
2. Bizga kerak oyna



3. Biz < Concole Application > ni tanlaymiz

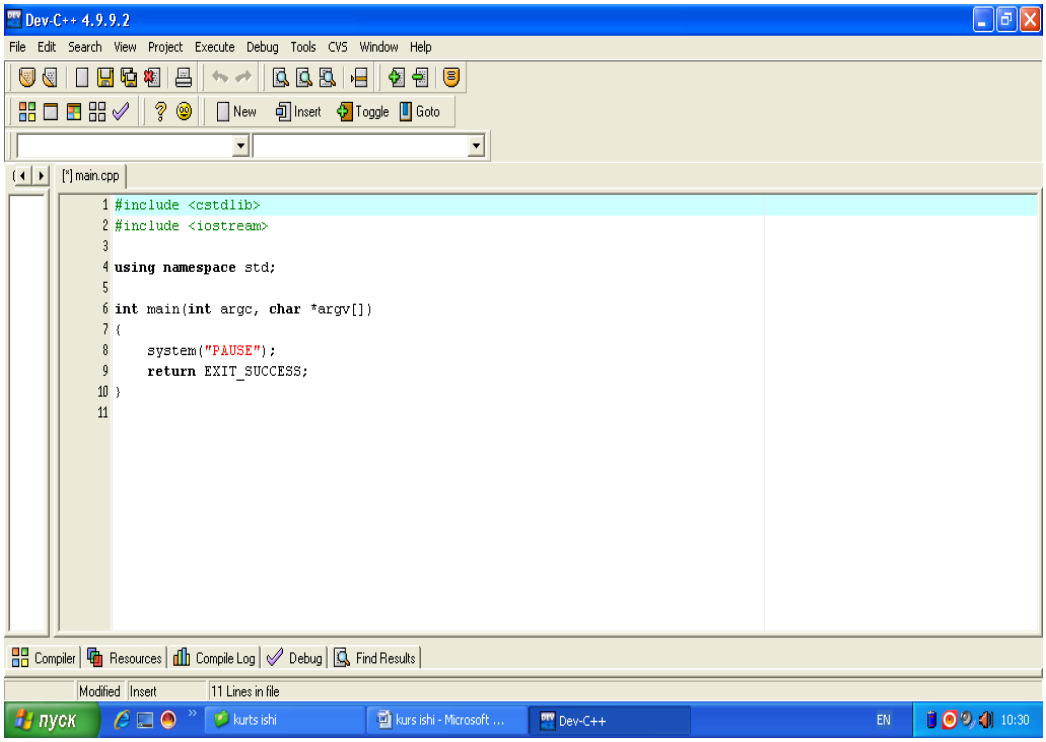


4. < Ok > bosganimizdan so’ng



5. Oynasi chiqadi. Ushbu oyna bizga tuzmoqchi bo’lgan dasturimiz qismlarini qayerga saqlashni va qanday nom bilan nomash imkoniyatlarini beradi.

< Сахранит > bossak



oynasi chiqadi.

6. Ushbu oynaning eng yuqori qismida dastur kutubxonalarini chaqirib olinadi

Biz tuzmoqchi bo'lgan dastur juda sodda bo'lib bizga uni tuzishda C++ matematik kutubxonasidan foydalanamiz.

Ko'rib turganingizdek biz C++ dasturlash tini ishga tushirib oldik.

{

belgisidan keyin dasturni kiritamiz.

7. Biz tuzgan dastur quyidagicha.

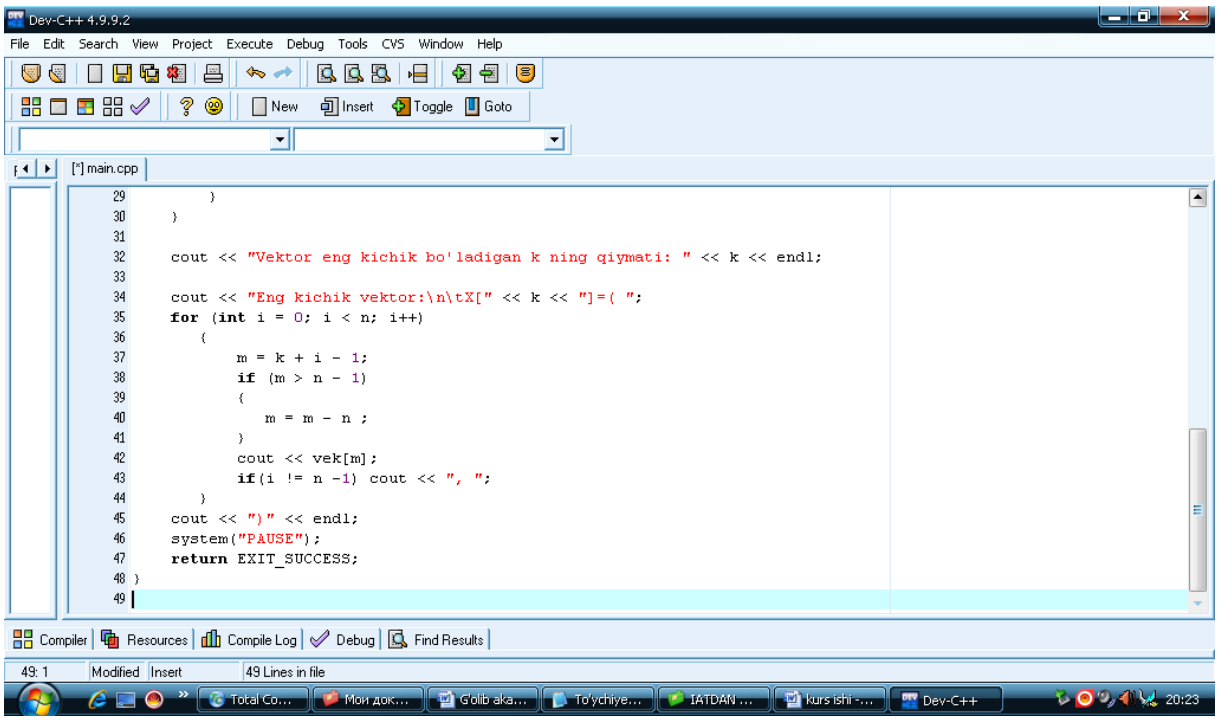
#include <cstdlib>

#include <iostream>

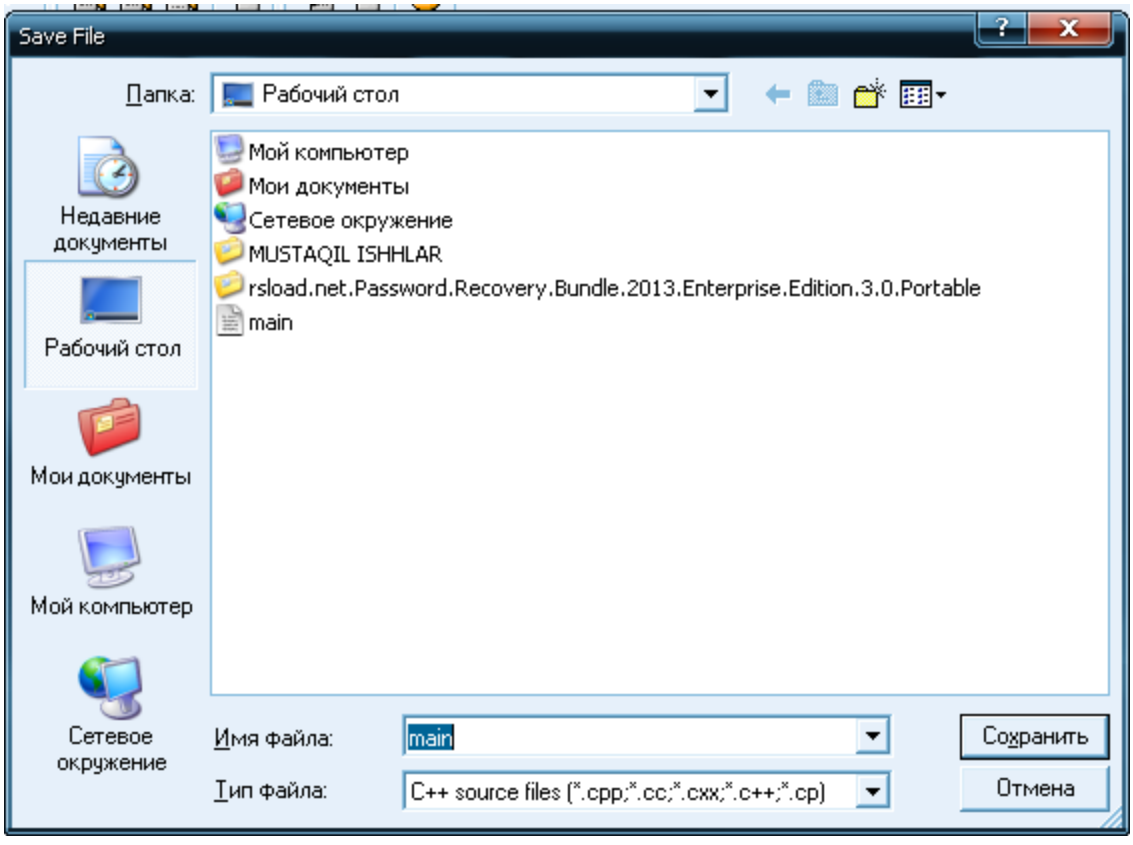
```
using namespace std;


int main(int argc, char *argv[])
{
   int vek[12];
   int n = 12;//aylana bo'lab yoziladigan sonlar soni
   int min;//eng kichik sonni aniqlash uchun
        int k;// eng kichik element turgan tartib raqam
        int m ;//qo'shimcha o'zgaruvchi


   cout<<"12 ta son kiriting\n";
   //Elementlarni kiritish
   for (int i = 0; i < n; i++)
   {
      cout << "vek[" << i + 1 <<"]=" ;
      cin >> vek[i];
   }
   min = vek[1];// vek[] nomli massivni birinchi elementini kichik deb olish
   // eng kichik elementni tanlash
   for (int i = 0; i < n; i++)
   {
      if (min > vek[i])
      {
         min = vek[i];
```

```
            k = i + 1;
        }
    }

    cout << "Vektor eng kichik bo'ladigan k ning qiymati: " << k << endl;

        cout << "Eng kichik vektor:\n\tX[" << k << "]=( ";
    for (int i = 0; i < n; i++)
      {
        m = k + i - 1;
        if (m > n - 1)
        {
          m = m - n ;
        }
        cout << vek[m];
        if(i != n -1) cout << ", ";
      }
    cout << ")" << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

8. Ko’rib turganingizdek ushbu dasturni tuzishda C++ dasturlash tilining massiv operatori imkoniyatlaridan faydalandik. Endi ushbu daturni C++ dasturlash tilida ishga tushiramiz.

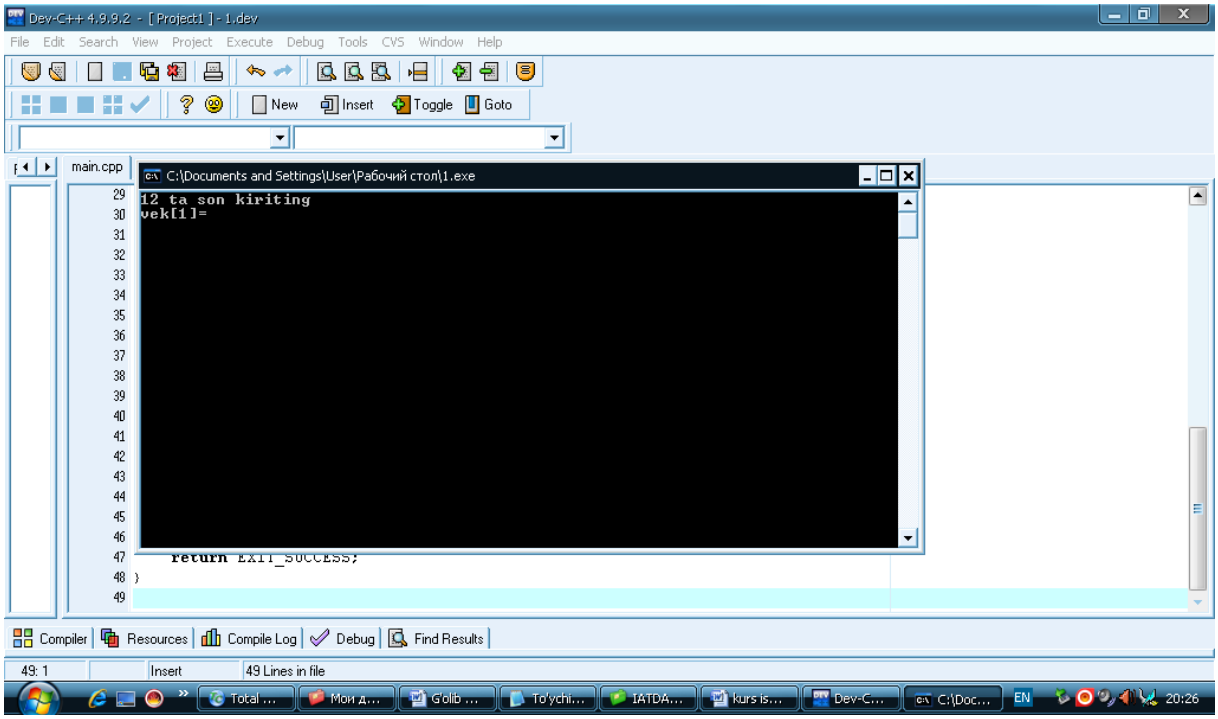


9. Daturni ishga tushirish uchun F9 tugmasini bosamiz.
10. So’ng bizga quyidagi oyna chiqadi.

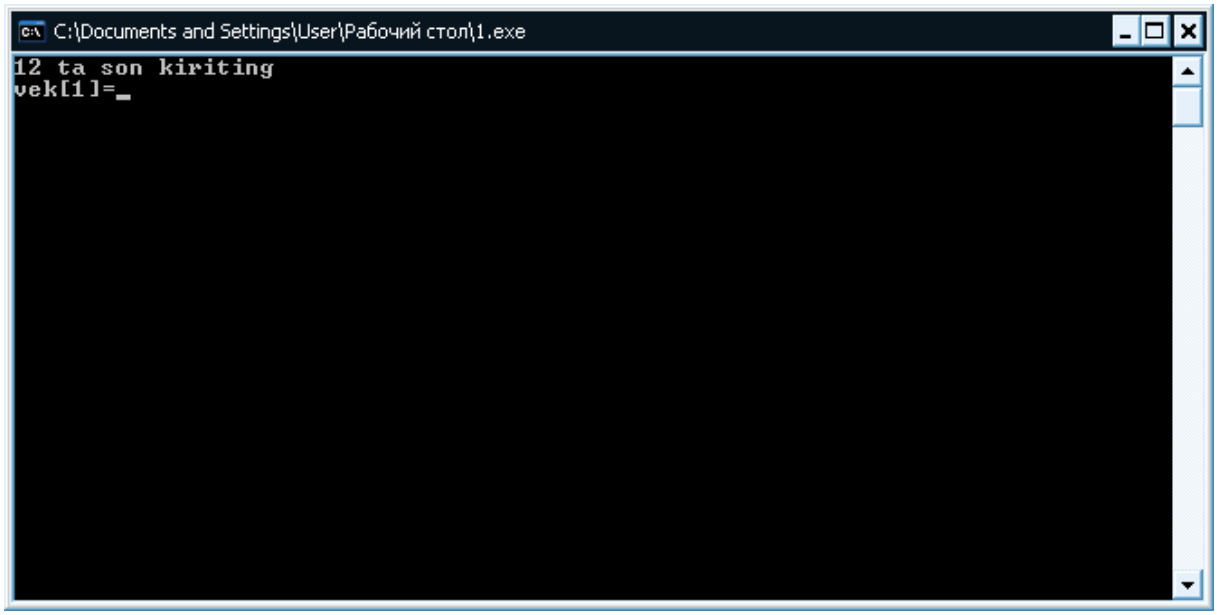


11. Ushbu oyna yordamida dasturning asosiy tana qismiga nom berish va dasturchi istagan joyga saqlab qo’yishi mumkin.

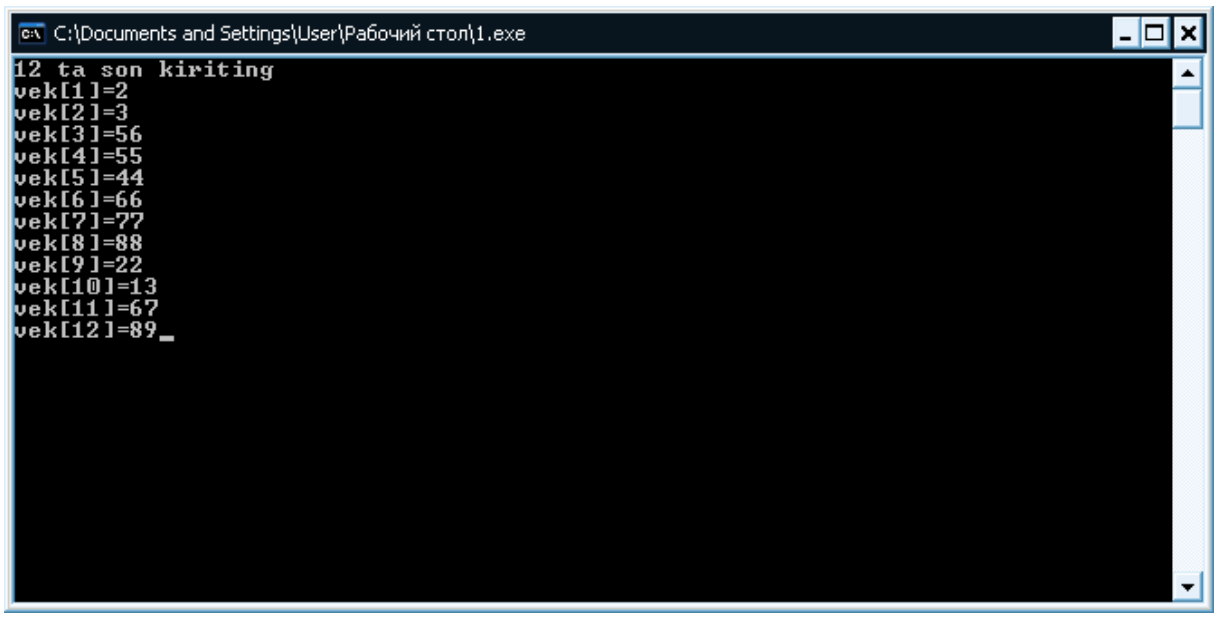
Bundan keyin dastur ishga tushadi va quyidagi oyna chiqadi



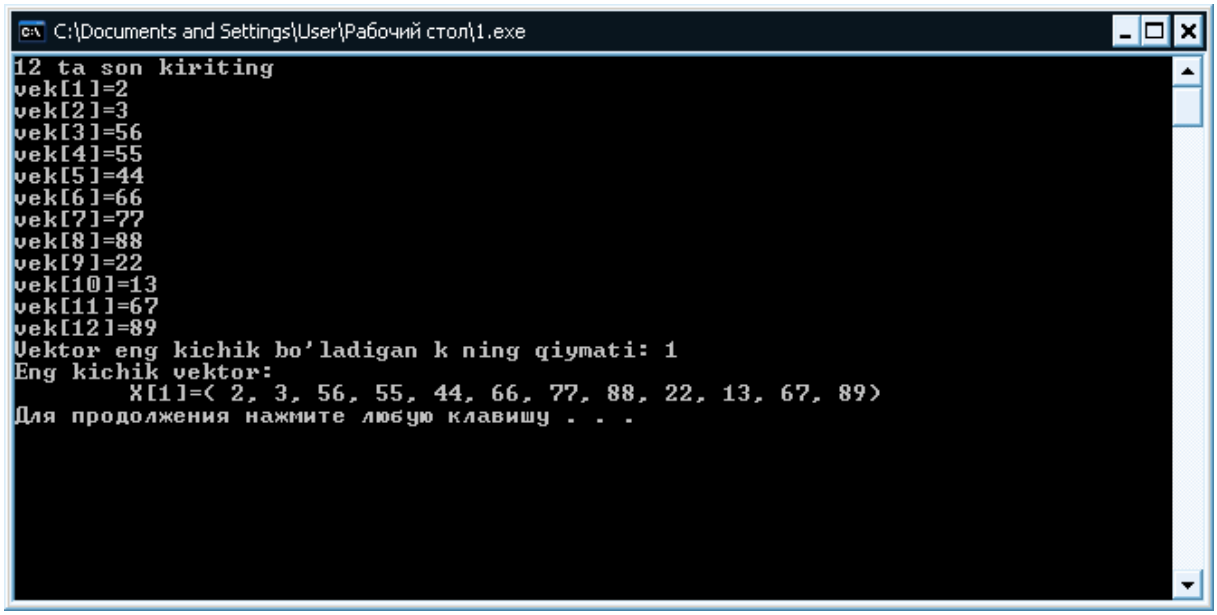
12. Ushbu ya’ni qora oynaga



13. Kerakli sonlarni ya’ni 12 ta son kiritamiz.



14.So’ralga barcha sonlarni kiritdik endi Enter tugmasini bosamiz



15.Ko’rib turganingizdek bizga eng kichik vektorni topib beradi.

Xulosa:

Men To’ychiev Axror ushbu Kurs ishini bajarish mobaynida eng avvalo dasturlash, haqida u nima o’zi dasturlashning qanday talablari va imkoniyatlari borligi haqida, songra dasturlash tillari va ularning bir-biridan farqlari va o’xshashlik tomonlari, kamchilik va afzalliklari haqida kerakli ma’lumotlarga ega bo’ldim.

Qolaversa men ushbu dasturni C++ dasturlash tilida tuzish davomida C++ dasturlash tili haqida o’z bilimlarimni yanada mustahkamlab oldim va qo’shimcha ma’lumotlarga ega bo’ldim.

Men tuzgan dasturda C++ dasturlash tilining massiv operatoridan foydalanganim tufayli men massivlar haqida ham ma’lumotlarga ega bo’ldim.

Foydalanilgan adabiyotlar

C++ dasturlash tili haqida ma’ruza 2011.

C++ dastur uz.

Dasturlash tillari haqida ma’ruza 2012.

Dasturlash haqida ma’ruza 2013.