23-MA’RUZA. DASTURLASHDA FAYLLAR BILAN ISHLASH. IFSTREAM SINFI

1. Oqim va bufer

C ++ kiritish va chiqarishni muhokama qilish murakkab jarayon hisoblanadi. Bir tomondan, deyarli har bir dastur kirish va chiqishdan foydalanadi va ulardan qanday foydalanishni o‘rganish dasturlash tilini o‘rganuvchilar duch keladigan birinchi muammolardan biridir.

Boshqa tomondan, C++ ushbu amallarni realizatsiya qilish uchun eng zamonaviy til xususiyatlaridan foydalanadi, jumladan sinflar, ishlab chiqilgan sinflar, funksiyalarni qayta yuklash, virtual funksiyalar, shablonlar va ko‘plab vorisxo‘rlik.

Ushbu mavzuda C++da kirish va chiqish sinflarini batafsil ko‘rib chiqamiz, ularning qanday tuzilganligini ko‘rsatadi va chiqish formatini qanday boshqarish kerakligini tushuntiradi.

ANSI/ISO C++ standartlashtirish qo‘mitasi C++ kiritish-chiqarishni mavjud C tilidagi kiritish-chiqarish bilan yanada moslashtirish uchun bir qator harakatlarni amalga oshirdi va bu an'anaviy C ++ yondashuvlariga nisbatan ba’zi o‘zgarishlarga olib keldi.

C ++da kiritish va chiqarishga tavsif. Ko‘pgina dasturlash tillarida kirish va chiqarish amallari tilning o‘zida o‘rnatilgan. Masalan, BASIC va Paskal kabi tillardagi kalit so‘zlar ro‘yxatini ko‘rib chiqsangiz, PRINT, writeln va shunga o‘xshash bayonotlar ushbu tillarning so‘z birikmalarining bir qismi ekanligini ko‘rasiz. Ammo C va C ++ da tilning o‘zida o‘rnatilgan kirish va chiqarish amallari mavjud emas. Agar ushbu tillarning kalit so‘zlarini ko‘rib chiqsangiz, masalan, for va if topasiz, ammo kiritish va chiqarish bilan bog‘liq hech narsa yo‘q. Dastlab, C tili kompilyator yozuvchilar ixtiyorida kiritish va chiqarishni qoldirdi.

Ushbu yondashuvning sabablaridan biri kompilyator ishlab chiquvchilariga maqsadli platformaning qurilma talablariga eng mos keladigan tarzda kiritish-chiqarish funksiyalarini loyihalashda bir oz erkinlik berish edi. Amalda, kiritish va chiqarish dasturlarining aksariyati dastlab Unix muhiti uchun ishlab chiqilgan kutubxona funksiyalariga asoslangan. ANSI C ushbu paketning texnik xususiyatlarini "Standart kirish / chiqish to‘plami" deb nomlangan standartda rasmiylashtirdi va uni C standart kutubxonasining ajralmas qismi sifatida o‘rnatdi. C++ bu to‘plamni ham taniydi, shuning uchun siz C oilasi bilan tanish bo‘lsangiz **stdio.h** faylida e'lon qilingan funksiyalar, ularni C++ dasturlarida ishlatishingiz mumkin. (Yangi xususiyatlar ushbu xususiyatlarni qo‘llab-quvvatlash uchun stdio sarlavhasidan foydalanadi.)

Biroq, kiritish-chiqarish uchun C++ dasturlash tili C yechimlari o‘rniga C++ yechimlariga tayanadi va bu yechim iostream (avvalgi iostream.h) va fstream

(avvalgi fstream.h) sarlavhalarida belgilangan sinflar to‘plamidir. Ma’lumot berilgan sinf kutubxonasi rasmiy til ta’rifiga kirmaydi (cin va istream kalit so‘zlar emas); chunki dasturlash tili sinflarni yaratish kabi vazifalarni bajarish qoidalarini belgilaydi, lekin bu qoidalarga amal qilish orqali aniq nimani yaratish kerakligini aniqlamaydi. Ammo C dasturlari standart funksional kutubxonalar bilan ta’minlanganidek, C++ standart sinf kutubxonalari bilan birga keladi. Dastlab standart sinf kutubxonasi norasmiy standart talablariga bo‘ysungan bo‘lib, u faqat iostream va fstream sarlavhalarida belgilangan sinflarni o‘z ichiga olgan. ANSI/ISO C++ standartlashtirish bo‘yicha qo‘mitasi ushbu kutubxonani standart sinf kutubxonasi sifatida rasmiylashtirishga qaror qildi va bir nechta standart sinflarni qo‘shishga qaror qildi. Ushbu mavzuda C++ standarti kiritish-chiqarishga e'tibor qaratildi.

Oqimlar va buferlar. C++ dasturi kirish va chiqishni baytlar oqimi sifatida ko‘rib chiqadi. Kirish paytida dastur kirish oqimidan baytlarni chiqaradi va chiqishda baytlarni chiqish oqimiga qo‘yadi. Matnli dasturlar uchun har bir bayt belgini anglatishi mumkin. Umuman olganda, baytlar belgilar va raqamli ma’lumotlarning ikkilik ko‘rinishini shakllantirishi mumkin. Kirish oqimi baytlari klaviaturadan, shuningdek qattiq disk kabi saqlash qurilmalaridan yoki boshqa dasturdan olinishi mumkin. Xuddi shu tarzda, chiqish oqimi baytlari ekranga, printerga, saqlash qurilmasiga yoki boshqa dasturga yuborilishi mumkin. Oqimlar dastur va oqim manbai yoki manzili o‘rtasida vositachilik vazifasini bajaradi. Ushbu yondashuv C++ dasturlariga klaviatura kiritilishini fayldan kirish bilan bir xilda ishlashga imkon beradi; C++ dasturi baytlar oqimini skanerlash orqali baytlarning qayerdan kelganligini bilishga hojat qoldirmaydi. Xuddi shu tarzda, oqimlardan foydalangan holda, C++ dasturi baytlarning haqiqatan ham qayerga yuborilganligidan qat'i nazar, chiqishni qayta ishlashi mumkin. Shunday qilib, kirishni boshqarish ikki bosqichni o‘z ichiga oladi:

• oqimni dastur kiritish bilan bog‘lash;

• oqimning faylga ulanishi.

Boshqacha qilib aytganda, kirish oqimi har bir tomondan ulanishni talab qiladi. Fayl tomonidagi ulanish oqim uchun ma’lumot manbai bo‘lib, dastur tomon ulanish oqim chiqishini dasturga yuklaydi. (Fayl tomonidagi ulanish fayl bo‘lishi mumkin, lekin u klaviatura kabi qurilma ham bo‘lishi mumkin.) Xuddi shunday, chiqishni boshqarish chiqish oqimini dasturga ulashni va ba’zi bir chiqish manzilini ushbu oqim bilan bog‘lashni o‘z ichiga oladi.

Odatda, kirish va chiqishni bufer orqali samaraliroq qayta ishlash mumkin. **Bufer** - bu qurilmadan dasturga yoki dasturdan qurilmaga ma’lumot uzatishda oraliq vaqtinchalik saqlash sifatida ishlatiladigan xotira blokidir.

Masalan, dastur qattiq diskda joylashgan fayldagi dollar belgilar sonini hisoblashi kerak. Dastur fayldan bitta belgini o‘qishi, uni qayta ishlashi, keyingi

belgini fayldan oʻqishi kabi va hokazo amallarni bajarishi kerak. Disk faylidan bir vaqtning oʻzida bitta belgini oʻqish juda koʻp qoʻshimcha qurilmalarga kirishni talab qiladi va sekin amalga oshiriladi.

Buferga asoslangan yondashuv asosida diskdan katta hajmdagi ma'lumotlarni oʻqish, bu qismni buferda saqlash va keyin buferdan birma-bir belgini oʻqish yotadi. Xotiradan alohida baytlarni oʻqish qattiq diskdan oʻqishga qaraganda ancha tezroq boʻlganligi sababli, ushbu yondashuv qurilmada sezilarli darajada tezroq va osonroq. Albatta, dastur bufer tugagandan soʻng, u keyingi ma'lumotlarni diskdan oʻqishi kerak. Ushbu tamoyil yomon ob-havo paytida koʻp miqdordagi yomgʻir suvini saqlaydigan suv omboridan foydalanishga oʻxshaydi, undan keyin suv kerak boʻlganda uyga kichik qismlar bilan oqadi. Xuddi shunday, dastur chiqarilayotganda dastur avval buferni toʻldirishi va keyin butun ma'lumotlar blokini qattiq diskka uzatishi, keyingi chiqishi uchun buferni tozalashi mumkin. Ushbu jarayon buferni tozalash deb nomlanadi.

Faylni kiritish va chiqarish. Koʻpgina kompyuter dasturlari fayllar bilan ishlaydi va shuning uchun fayllarni yaratish, oʻchirish, yozish, oʻqish, ochish kerak boʻladi.

Fayl - bu qurilmalarda saqlanadigan nomlangan baytlar toʻplami. Fayl ma'lum bir baytlar ketma-ketligi sifatida tushunilishi kerak, bu oʻziga xos nomga ega, masalan, dastur.txt fayli. Xuddi shu nomdagi fayllar bitta katalogda boʻlishi mumkin emas. Fayl nomi nafaqat uning nomini, balki uning kengaytmasini ham anglatadi, masalan: file.txt va file.dat bir xil nomlarga ega boʻlsa-da, har xil fayllardir. Fayllarning toʻliq nomi kabi tushuncha mavjud - bu fayl nomi bilan fayl katalogiga toʻliq manzil, masalan: D:\docs\file.txt. Ushbu asosiy tushunchalarni tushunish juda muhim, aks holda fayllar bilan ishlash qiyin boʻladi.

Fayllar bilan ishlash uchun **<fstream>** sarlavha faylini kiritishingiz kerak. **<fstream>** bir nechta sinflarni belgilaydi va <ifstream> - fayl kiritish va <ofstream> - fayl chiqishi sarlavha fayllarini oʻz ichiga oladi.

Faylni kiritish/chiqarish standart kiritish/chiqarishga oʻxshaydi, faqat farq shundaki, kiritish/chiqarish ekranga emas, balki faylga amalga oshiriladi. Agar standart qurilmalarga kiritish-chiqarish **cin** va **cout** moslamalari yordamida amalga oshirilsa, u holda fayllarni kiritish-chiqarishni tartibga solish uchun cin va cout operatorlariga oʻxshash ishlatilishi mumkin boʻlgan oʻz obyektlarini yaratish kifoya.

Masalan, matnli fayl yaratishingiz va unga C++ dagi fayllar bilan ishlash satrini yozishingiz kerak. Buning uchun quyidagi amallarni bajarishingiz kerak:

1) oqim sinfining obyektini yaratish;
2) sinf obyektini yozilishi kerak boʻlgan fayl bilan bogʻlash;
3) faylga satr yozish;
4) faylni yopish.

Nima uchun **ifstream** sinfini emas, balki **ofstream** sinfining obyektini yaratish kerak? Siz faylga yozishingiz kerak, agar siz fayldan ma’lumotlarni o‘qishingiz kerak bo‘lsa, **ifstream** sinfining obyekti yaratiladi.

Faylga yozish uchun obyekt yaratish:

ofstream obyekt_nomi;

Masalan, obyekt nomi fout bo‘lsa, quyidagicha bo‘ladi:

ofstream fout

Nima uchun ushbu obyekt bizga kerak? Obyekt faylga yozish imkoniyatiga ega bo‘lishi kerak. Obyekt allaqachon yaratilgan, ammo satr yozilishi kerak bo‘lgan fayl bilan bog‘liq emas.

fout.open("dastur.txt"); //obyektni faylga bog‘lash

Nuqta amali orqali open()metodiga ega bo‘lamiz, qavs ichida fayl nomini ko‘rsatamiz. Belgilangan fayl dastur bilan joriy katalogda yaratiladi. Agar ushbu nomdagi fayl mavjud bo‘lsa, mavjud fayl yangisi ustiga yoziladi. Shunday qilib, fayl ochiq, unga kerakli qatorni yozish qoladi. Bu quyidagicha amalga oshiriladi:

fout <<"Faylga ushbu satr qo‘shiladi";

fstream sarlavhasi fayldan ma’lumotlarni o‘qish va faylga yozish uchun funksionallikni ta’minlaydi. Umuman olganda, u konsol bilan ishlaydigan iostream sarlavhasiga juda o‘xshaydi, chunki konsol ham fayldir. Shuning uchun, barcha asosiy amallar deyarli bir xil.

Eng keng tarqalgan amallar quyidagilar:

1) Kiritish-chiqarishni qayta yo‘naltirish operatorlari - << va >>
2) getline() va get() bilan put () satrlarni yozish va o‘qish usullari
3) write() va read() metodlari orqali oqimga yozish va o‘qish
4) open() va close() fayllarini ochish /yaratish va yopish metodlari
5) Faylning is_open() ochiqligini va faylning oxiriga yetganligini tekshirish eof() metodlari
6) Formatlangan chiqishni width() va precision() bilan >> uchun sozlash

Bu oqim kutubxonasi taqdim etadigan barcha xususiyatlar emas. Yuqorida keltirilgan metodlar bilan tanishib chiqamiz.

2. Faylni o’qish. ifstream sinfi

ifstream - ushbu ma’lumotlar turi kirish fayllari oqimini ifodalaydi va fayllardan ma’lumotlarni o‘qish uchun ishlatiladi.

Ushbu sinf fayllarni o‘qish imkoniyatini beradi. Faylni ochishning ikki yo‘li mavjud: open() metodini chaqirish yoki konstruktorda unga yo‘lni ko‘rsatish. Kodni yozishni boshlashdan oldin matnli faylni tayyorlashingiz kerak. D diskida dastur nomli papkani yarating va unda txt kengaytmasi bilan “dasturmisol” faylini yarating.

```
#include <iostream>
#include <fstream> // kutubxonani ulash
using namespace std;

int main()
{
  ifstream file; // ifstream obyektining sinfini hosil qilish
  file.open("d:\\dastur\\dasturmisol.txt"); // faylni ochish
}
```

Faylni konstruktorda ochish quyidagicha:

```
#include <iostream>
#include <fstream> // kutubxonani ulash
using namespace std;

int main()
{
  ifstream file ("d:\\dastur\\dasturmisol.txt"); // faylni konstruktorda ochish
}
```

Bu dasturda d diskda joylashgan dastur nomli papkadagi dasturmisol.txt nomli txt faylini ochish so‘ralmoqda.

open() metodidan foydalanish dasturchi darhol faylga murojaat qilishni istamasa, qulay metoddir.

Agar ma’lum bir funksiya ichida faylni ochish, u bilan ishlash va uni yopish kerak bo‘lsa, unda faylga yo‘lni to‘g‘ridan-to‘g‘ri konstruktorda yozishingiz mumkin. Umuman olganda, bu vaziyatga bog‘liq.

Faylni ochgandan so‘ng, uning ochilgan yoki ochilmaganini aniqlashga to‘g‘ri keladi. Fayl ochilmasligi uchun bir nechta sabablar bo‘lganligi sababli, uni ochilgan yoki ochilmaganini darhol aytolmaymiz. Masalan, ko‘rsatilgan nomga ega fayl belgilangan papkada yo‘q yoki yo‘l noto‘g‘ri ko‘rsatilgan bo‘lsa, fayl ochilmaydi. Siz ikki yo‘l bilan borishingiz mumkin: fayl o‘zgaruvchisini mantiqiy ifodada tekshirish (masalan, "!" operatoridan foydalanish orqali) yoki **is_open()** metodidan foydalanish orqali:

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

int main()
{
  ifstream file ("d:\\dastur\\dasturmisol.txt");

  if (!file)
  {
        cout << "Fayl ochilmadi\n\n";
        return -1;
  }
  else
  {
        cout << "Fayl ochildi!\n\n";
        return 1;
  }
}
```

is_open() metodi yordamida tekshirish uchun ikkinchi variant quyidagicha bo‘lishi mumkin:

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

int main()
{
```

```
  ifstream file ("d:\\dastur\\dasturmisol.txt");

  if (file.is_open()) //  is_open() metodini chaqirish
        cout << "Fayl ochildi!" << endl;
  else
  {
        cout << "Fayl ochilmadi!" << endl;
        return -1;
  }
}
```

Agar fayl topilsa va muvaffaqiyatli ochilsa, is_open () usuli 1 ni qaytaradi. Aks holda, u 0 ni qaytaradi va else blokida yozilgan kod ishlaydi.

Agar fayl ochiq bo‘lmasa, xatoni ko‘rib chiqish tavsiya etiladi. Qoida tariqasida, agar dasturning barcha ishlari fayl bilan bog‘liq bo‘lsa, ular konsolga xabar yozadilar va dasturdan chiqadilar. Jiddiy xatolar bo‘lsa, u yoki bu xatoni tavsiflaydigan ma’lum bir ijro kodini (raqamini) qaytarish odatiy holdir. Dastur muallifi har bir xato turi uchun o‘z kodlarini o‘ylab topishi mumkin.

Agar fayl muvaffaqiyatli ochilgan bo‘lsa, siz undan ma’lumotni o‘qishingiz mumkin.

>> o‘qish operatori

iostream kutubxonasidagi kabi o‘qishni >> operatori bilan tashkil qilish mumkin, bu qaysi o‘zgaruvchining o‘qilishini bildiradi:

```
double d;
int i;
string s;

file >> d >> i >> s;
```

Haqiqiy, butun son va satrni o‘qiydi. Agar satr oxiri yoki bo‘sh joy (probel) paydo bo‘lsa, satrni o‘qish tugaydi. Shuni ta’kidlash kerakki, >> operatori matnli fayllarga taalluqlidir. Ikkilik fayldan o‘qishning eng yaxshi usuli **read()** metodi yordamida amalga oshiriladi.

Agar faylni so‘zlarga bo‘lishni amalga oshirmoqchi bo‘lsak, ushbu operator juda qulaydir:

```
// fayldan so‘zlarni o‘qish
for(file >> s; !file.eof(); file >> s)
        cout << s << endl;
```

getline() va get() metodlari. Bitta qatordagi satrni to‘liq o‘qish **getline**() metodi yordamida iostreamda bo‘lgani kabi amalga oshiriladi. **string** turidagi satr o‘qilgan bo‘lsa, uning qayta aniqlangan muqobilini funksiya sifatida ishlatish tavsiya etiladi:

```
//matndan satrni o‘qish
#include <iostream>
#include <fstream>
using namespace std;

int main()
{

   ifstream file ("d:\\accounts.txt");
   string s;
   getline(file, s);
   cout << s << endl;

}
```

Masalan, accounts.txt faylida quyidagi ikkita satr berilgan bo‘lsin:

```
Bu birinchi satr
Bu ikkinchi satr
```

Bu holda ekranda “Bu birinchi satr” yozuvi chiqadi. Ya’ni string tipidagi s satrining qiymati Bu birinchi satr satri bo‘ladi.

Agar sizga **char[]** belgilar qatorini o‘qish kerak bo‘lsa, u holda **get()** yoki **getline()** ni xuddi shunday metod sifatida o‘qish mumkin:

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

int main()
{
```

```
   ifstream file ("d:\\accounts.txt");
          int n = 10;
        //O‘qish uchun bufer yaratish
  char* buffer = new char[n+1];
       buffer[n]=0;
//n ta simvolni o‘qish
  file.get(buffer,n);
//Yoki birinchi bo‘sh joygacha (probelgacha) o‘qish
  file.getline(buffer,n,' ');
//o‘qilgan belgilarni chop etish
  cout << buffer;
//buferni bo‘shatish
  delete [] buffer;

}
```

read() metodi.

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

int main()
{

   ifstream file ("d:\\accounts.txt");
   //Fayldan N baytni o‘qish
  int n=10;
//Bufer yaratish
  char* buffer=new char[n+1]; buffer[n]=0;
//Baytlarni o‘qish
  file.read(buffer,n);
//Ekranda chop etish
  cout<<buffer;
  delete [] buffer;
}
```

Bu xuddi oldingi misol kabi. Faqatgina bittagina holat mavjud, ya’ni ajratuvchini aniqlay olmaysiz. **read()** formatlanmagan kiritish uchun ishlatiladi.

Birinchi navbatda ikkilik fayllarni o‘qish uchun mo‘ljallangan. Matnli fayl binar faylning xususiy holi bo‘lgani uchun, bu usul matnli faylga ham tegishli.

close() metodi. Faylni yopadi. Boshqa hech qanday vazifasi mavjud emas. Agar faylni yopmasdan dasturdan chiqsangiz, o‘qish uchun ochilgan fayl buzilishi juda kam holatlarda uchraydi. Ammo shuni aytish kerakki, ochilgan faylning yopilishi eng yaxshi holatdir.

eof() metodi. Faylning oxiriga yetib borganligini tekshiradi. >> operatori yordamida so‘zlarni o‘qish bilan bog‘liq yuqoridagi misol bunday tekshiruvdan foydalanadi**.**

seekg() metodi. Kiritilgan raqam orqali joriy pozitsiyani o‘rnatadi. Ushbu metodga pozitsiyalash usuli jo‘natiladi:

- ios_base::end – faylning oxiridan yangi pozitsiyani sanash
- ios_base::beg - faylning boshidan yangi pozitsiyani sanash
- ios_base::cur - faylning joriy holatidan n bayt o‘tish (jimlik bo‘yicha)

```
        file.seekg(0,ios_base::end); //faylning oxiridan hisoblash
    file.seekg(10,ios_base::end); //oxiridan 10 bayt hisoblash
    file.seekg(30,ios_base::beg); //31 bayt hisoblash
    file.seekg(3,ios_base::cur); //3 bayt tashlab hisoblash
    file.seekg(3); //yuqorida ko‘rinishning analogi - 3 bayt tashlab hisoblash
```

tellg() metodi. Ba’zi holatda qancha miqdordagi ma’lumot o‘qilganligi haqida ma’lumot olish kerak bo‘ladi. Bunga tellg () metodi yordam beradi:

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

int main()
{

  ifstream file ("d:\\accounts.txt");
    string s;
    getline(file, s);
    cout<<"O‘qilgan bayt: " <<file.tellg();
}
```

Bu **int** tipidagi qiymatni qaytaradi, bu esa qancha baytda o‘qilganligini ko‘rsatadi. Fayl hajmini olish uchun u seekg() metodi bilan birgalikda ishlatilishi mumkin:

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

int main()
{

  ifstream file ("d:\\accounts.txt");
    string s;
  file.seekg(0,ios_base::end);
    cout<<"O‘qilgan bayt: " <<file.tellg();
}
```