# **Киргизүү/чыгаруунун кошумча мүмкүнчүлүктөрү**

## **Колдонуучулар манипуляторлорун түзүү**

Киргизүү/чыгаруу операторлорун кайрадан жүктөөгө толуктоо катары, колдонуучу өзүнүн манипуляторлорун аныктаса болот. Мындай манипуляторлорду колдонуунун эки маанилүү жагы бар. Биринчиден, удаалаш жайгашкан бир нече киргизүү/чыгаруу амалдарды бир манипуляторго бириктирсе болот. Экинчиден, манипулятор киргизүү/чыгаруу процессин стандарттуу эмес жабдууларды колдонууда кереги тиет.

Манипуляторлордун эки база тиби бар: киргизүү агымдар менен иштеген жана чыгаруу агымдары менен иштеген. Булардан сырткары дагы эки категория бар: аргументи бар жана аргументи жок манипуляторлор.

Чыгаруу үчүн колдонгон параметрсиз манипулятор төмөнкү түзүлшкө ээ:

ostream манипулятор\_ысымы(ostream &агым)  
{//манипулятордун коду  
return агым;  
}

Киргизүү үчүн колдонгон параметрсиз манипулятор төмөнкү түзүлүшкө ээ:

istream &манипулятор\_ысымы(istream &агым)  
{//манипулятордун коду  
return агым;  
}

Жөнөкөй мисалга назар салалы. Бул мисалда *misal* манипулятору түзүлүп, чыгаруу үчүн талаанын узундугун 12 белги, тактыгын 5 жана толуктоо белгиси катары \* символун аныктайт.

#include <iostream>  
using namespace std;  
  
ostream &misal(ostream &stream)  
{  
stream.width(12);  
stream.precision(5);  
stream.fill('\*');  
return stream;  
}  
  
int main()  
{  
cout<<misal<<123.456789;  
return 0;  
}

Жыйынтык катары төмөнкүнү алабыз:

\*\*\*\*\*\*123.46

Колдонуучу манипуляторлорду көп колдонгон сөз же сүйлөмдөрдү чыгаруу үчүн колдонсо болот. Мисалы,

#include <iostream>  
using namespace std;  
  
ostream &misal1(ostream &stream)  
{  
stream<<"Көңүл бургула: ";  
return stream;  
}  
  
ostream &misal2(ostream &stream)  
{  
stream<<"унутпагыла!";  
return stream;  
}  
  
int main()  
{  
cout<<misal1<<"абдан шыкалган ток!\n";  
cout<<"Жарыкты өчүргөндү "<<misal2;  
return 0;  
}

Програм төмөнкү жыйынтыкты берет:

Көңүл бургула:  абдан шыкалган ток!  
Жарыкты өчүргөндү унутпагыла!

Төмөнкү мисалда манипуляторду туура паролду киргизүү үчүн колдонобуз.

#include <iostream>  
#include <cstring>  
using namespace std;  
  
istream &misal(istream &stream)  
{  
cout<<'\a';//динамиктин үнү  
cout<<"Паролду киргиз:";  
return stream;  
}  
  
int main()  
{char parol[20];  
    do{  
        cin>>misal>>parol;  
    }while (strcmp(parol,"123"));  
    cout<<"Парол туура болду\n";  
return 0;  
}

## **Файлга жазуу жана файлдан окуу**

Файлдык жана консолдук киргизүү/чыгаруу өз ара тыгыз байланышта турат. Ошондуктан жогоруда айтылган маалыматтарды файлдарга карата колдонсо болот.

Файлга жазуу жана файлдан окуп алуу үчүн програмдын башына **<***fstream***>** сабын кошуу керек. Бул файлда бир нече класстар аныкталган: *ifstream***,***ofstream***,***fstream***.**Ал класстар *istream*жана *ostream* класстарынын туунду класстары болуп саналышат.

Си++ тилинде файлды ачуу үчүн аны агым менен байланыштырат. Агымдын үч түрү бар: жазуу, окуп алуу жана жазуу/окуу. Файлды ачуу үчүн агым түзүү керек. Жазуу агымды түзүү үчүн *ofstream* тибиндеги объекти жарыялоо керек. Окуу агымын түзүү үчүн  *ifstream* тибиндеги объекти жарыялоо талап кылынат. Эгерде жазуу жана окуу агымдары бир мезгилде колдонулса, анда объекти *fstream*тибинде жарыялоосу зарыл. Төмөнкү мисалда бир агым жазуу үчүн, бир агым окуу үчүн жана бир агым жазуу/окуу үчүн түзүлөт:

ifstrteam in; //окуу  
ofstream out; //жазуу  
fstream io;//окуу - жазуу

Түзүлгөн агымды файл менен *open()* функция аркылуу байланыштырат. Бул функция аталган үч класстын мүчөсү болот. Төмөндө ал функциянын прототиби ар бир класс үчүн көрсөтүлгөн:

void ifstream::open(const char \*файл\_ысымы, openmode режим=ios::in);  
void ofstream::open(const char \*файл\_ысымы, openmode режим=ios::out | ios::trunc);  
void fstream::open(const char \*файл\_ысымы, openmode режим=ios::in | ios::out);

Мында ***файл\_ысымы*** маалымат сакталуучу же окуп алынуучу файлдын ысымы. Ал өз ичинде файлдын толук жолун (спецификаторун) камтышы мүмкүн. ***Режим*** төмөнкүлөрдүн бири болушу зарыл:

ios::app  
ios::ate  
ios::binary  
ios::in  
ios::out  
ios::trunc

Бул маанилердин бир нечесин OR оператору аркылуу бириктирсе болот.

*ios::app* мааниси берилиштерди файлдын айагына жазуу режиминде ачат. Бул маани жазуу үчүн гана ачылган файлдар үчүн колдонулат. *ios::ate* мааниси файлды ачуу убактысында файлдын айагын издөө режиминде колдонулат. *ios::binary* режими файлын колдонуу экилик режимде ачуу учурунда пайдаланылат. Унчукпаган режимде файлдар текст режиминде ачылат. *ios::in* режими файлга жазуу режиминде ачат. *ios::out* режими файлды окуп алуу режиминде ачат. *ios::trunc* режими файлды ачуу учурунда ичиндеги берилиштерди өчүрүп, файлдын узундугун нөлгө алып келет. *ofstream*кызматчы сөзү аркылуу чыгаруу агымын түзүү учурунда көрсөтүлгөн ысымдагы файл нөлдук узундукка чейин кесилет.

Төмөнкү мисалда *proba* файлы жазуу үчүн ачылат:

ofstream agym;  
agym.open(“proba”);

Эгерде *open()* функциясынын аткарылышы ката менен айактаса, анда логик туйунтмада агым *false* мааниге ээ болот. Бул фактыны файлдын туура ачылышын текшерүү үчүн колдонсо болот:

if(agym)  
{ cout<<”Файл ачык\n”; else cout<<”Файлды ачууга мүмкүн эмес”;}

Файлдын туура ачылышын *ifstream***,***ofstream*жана *fstream* класстарынын мүчөсү болгон *is\_open()* функция аркылуу текшерсе болот. Төмөндө ал функциянын прототиби көрсөтүлгөн:

bool is\_open();

Эгерде агым ачык файл менен туура байланыштырылса, функция чын деген маани берет, болбосо жалган деген мааниге ээ болобуз. Төмөндө *fail* агымы менен байланыштырылган файл ачылгандыгы текшерилет:

if(agym.is\_ope())  
{cout<<”Файл ачык\n”; else cout<<”Файл ачык эмес\n”;}

Файлдын ачылган же ачылбаганын текшерүү үчүн *is\_open()* функциясын колдонуу бир четинен туура. Бирок *ifstream***,***ofstream*жана *fstream* класстарында файлдарды автоматтык түрдө ачкан конструкторлор бар. Ошондуктан файлды ачууда төмөнкү түзүмдү колдонуу ыңгайлуу болот:

ifstream agym(“proba”); //жазуу үчүн файлды ачуу

Файлды жабуу үчүн*close()* мүчө-функция колдонот. Бул функция параметр жана кайтарып берүүчү мааниге ээ болбойт. Мисалы, *agym* агымы менен байланышкан файлды жабуу үчүн төмөнкү көрсөтмө зарыл:

agym.close();

*ios*классынын мүчөсү болгон *eof()*функциянын жардамы менен файлдын айагына жеткендиги же жетпегендиги текшерилет. Файлдын айагына жеткен учурда функция *чын*, жетпеген учурда *жалган* деген маанини кайтарат. Төмөндө бул функциянын прототиби келтирилген:

bool eof();

Файл ачылгандан кийин ага текстик берилиштерди сактап же андан окуп алса болсо. Ал үчүн *cin* жана *cout* агымдарын файл менен байланышкан агымдарга алмаштырып **>>** жана **<<**операторлорун колдонобуз.

Төмөнкү програмда жазуу үчүн файл түзүлүп ал файлга берилиштер жазылат. Андан кийин файл жабылат дагы кайрадан окуп алуу үчүн ачылып ичиндеги берилиштер окулат.

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main()  
{  
    ofstream saktoo("misal");//жазуу үчүн файлды түзүү  
    if(!saktoo)  
    {cout<<"Файлды түзүүгө мүмкүн эмес\n";  
        return 1;  
    }  
    saktoo<<"Салам\n";  
    saktoo<<10<<' '<<100<<endl;  
    saktoo.close();  
    ifstream okup\_aluu("misal");//окуп алуу үчүн файлды ачуу  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    char sap[20];  
    int i,j;  
    okup\_aluu>>sap>>i>>j;  
    cout<<sap<<' '<<i<<' '<<j<<endl;  
    okup\_aluu.close();  
    return 0;  
}

Програм төмөнкү жыйынтыкты берет:

Салам 10 100

Төмөнкү мисалда сап баскычтоптон киргизилет дагы окулган саптар файлга сакталат. Програм баскычтоптон **!** белгиси киргизилгенде токтойт. Програмды колдонуу үчүн буйрук сапта сактоо үчүн түзүлүп жаткан файлдын ысымын бергиле.

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
    if(argc!=2)  
    {  
        cout<<"Файлдын ысымын киргизгиле\n";  
        return 1;  
    }  
  
    ofstream saktoo(argv[1]);//жазуу үчүн файлды түзүү  
    if(!saktoo){  
        cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    char sap[20];  
    cout<<"Сап киргизгиле.\n";  
    cout<<"Киргизүүнү токтотуу үчүн ! белгисин киргизгиле\n";  
    do{  
        cout<<": ";  
        cin>>sap;  
        saktoo<<sap<<endl;  
    }while(\*sap!='!');  
    saktoo.close();  
    return 0;  
}

Төмөнкү програмда файлдын көчүрмөсү алынат. Көчүрмө алуу учурунда файлдагы боштук белгилери **$** белгисине алмашытырылат.

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
    if(argc!=3)  
    {  
        cout<<"Файлдын көчүрмөсүн алуу\n";  
        return 1;  
    }  
  
    ifstream okup\_aluu(argv[1]);//окуп алуу үчүн файлды ачуу  
    ofstream saktoo(argv[2]);//жазуу үчүн файлды түзүү  
    if(!okup\_aluu){  
        cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    if(!saktoo){  
        cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
     
    char belgi;  
    okup\_aluu.unsetf(ios::skipws);  
    while(!okup\_aluu.eof())  
    {  
        okup\_aluu>>belgi;  
        if(belgi==' ')belgi='$';  
            if(!okup\_aluu.eof()) saktoo<<belgi;  
            }  
            okup\_aluu.close();  
    saktoo.close();  
    return 0;  
}

*misal* файлында төмөнкү саптар бар болсун дейли:

сап клавиатурадан киргизилет  
 дагы, окулган  
саптар файлга жазылат.

Бул програмды *proba* ысымы менен компилеп, буйрук саптан төмөнкү удаалаштыкты киргизсек:

proba misal misal1

анда *misal1* файлында төмөнкү саптар жазылат:

сап$ клавиатурадан$киргизилет  
$дагы,$окулган$  
саптар$файлга$жазылат.

## **Экилик калыпта киргизүү/чыгаруу**

ASCII кодунда сакталган тексттик файлдар көпчүлүк учурларда пайдалуу болсо дагы, аларда экилик файлдарындай ийкемдүүлүк жок. Си++тилинде экилик файлдар үчүн кеңири мүмкүндүктөр болгон сактоо/окуу функциялар колдонот. Ал функциялар сактоо/окуу процесстерин текшерүү үчүн мүмкүндүк беришет.

Экилик сактоо/окуунун төмөнкү деңгелинде *get()*жана *put()* функциялары орун алган. *put()* мүчө-функциясы аркылуу байтты жазса болот, ал эми *get()* функциясы аркылуу байтты окуп алса болот. *get()*жана *put()* функциялары бир катар калыпка ээ. Төмөндө эң көп кезиккен калыбы келтирилген:

istream &get(char &белги);  
ostream &put(char &белги);

*get()* функциясы аны менен байланышта турган агымдан бир белгини окуп алып аны *белги* аргументине берет. Маани катары агымга шилтемени кайтарып берет. Файлдын айагы белгисин окуп алган учурда функция *false* маанисин кайтарып берет. *put()* функциясы *белги*ни агымга жазып, кайрадан агымга шилтемени берет.

Экилик берилиштердин блогун окуу жана жазуу үчүн *read()* жана *write()* функциялары колдонулат. Төмөндө алардын прототиптери келтирилген:

istream &read(char \*буфер, streamsize байттын\_саны);  
ostream &write(char \*буфер, streamsize байттын\_саны);

*read()* функциясы агымдан *байттын\_саны*аргументинде көрсөтүлгөн сандагы байтты окуп алат дагы, аны *буфер*көрсөткүчү аныктаган буферге жиберет. *write()* функциясы *байттын\_саны*аргументинде көрсөтүлгөн сандагы байтты тиешелүү агымга жазат. *streamsize* типтин мааниси кандайдыр бир бүтүн типти аныктайт.

Эгерде *байттын\_саны а*ргументинде көрсөтүлгөн сандагы байттар окулбай файлдын айагына жетип калса, анда *read()* функциясы өз ишин айактайт дагы, буферде файлда бар болгон белгилер гана болот. Файлдан канча белги окуп алынганын билүү үчүн *gcount()* функциясы колдонулат. Бул функциянын прототиби төмөнкүдөй:

streamsize gcount();

Төмөнкү програм файлдын мазмунун экранга чыгат:

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
    char belgi;  
    if(argc!=2)  
    {  
        cout<<"Файлдын мазмуну\n";  
        return 1;  
    }  
    ifstream okup\_aluu(argv[1],ios::in | ios::binary);  
     if(!okup\_aluu){  
        cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
     
    while(!okup\_aluu.eof())  
    {  
        okup\_aluu.get(belgi);  
        cout<<belgi;}  
            okup\_aluu.close();  
     
    return 0;  
}

Төмөнкү мисалда колдонуучу тарабынан киргизилген белгилерди файлга жазуу үчүн *put()* функциясын колдонобуз.

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
    char belgi;  
    if(argc!=2)  
    {  
        cout<<"Файлга жазуу\n";  
        return 1;  
    }  
  
    ofstream saktoo(argv[1],ios::out | ios::binary);  
     if(!saktoo){  
        cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    cout<<"Токтотуу үчүн ! белгисин киргизгиле\n";  
    do{  
        cout<<":";  
        cin.get(belgi);  
        saktoo.put(belgi);  
    }while(belgi!='!');  
    saktoo.close();  
    return 0;  
}

Кийинки програм сап жана *double*тибиндеги санды жазуу үчүн*write()* функциясы колдонулат.

#include <iostream>  
#include <fstream>  
#include <cstring>  
using namespace std;  
  
int main()  
{  
    ofstream saktoo("Misal", ios::out | ios:: binary);  
    if(!saktoo)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    double san=156.78;  
    char sap[]="Бул сап";  
    saktoo.write((char \*) &san, sizeof(double));  
    saktoo.write(sap,strlen(sap));  
    saktoo.close();  
    return 0;  
}

*write()* функциясынын буфери белгилердин массиви катары жарыяланбагандыктан, ал функцияга кайрылуу учурунда типти (char \*) тибине келтирдик.

Бул мисалда түзүлгөн файлды берилиштерди окуу үчүн төмөнкү програм *read()* функцияны колдонот.

#include <iostream>  
#include <fstream>  
#include <cstring>  
using namespace std;  
  
int main()  
{  
    ifstream okup\_aluu("Misal", ios::in | ios:: binary);  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    double san;  
    char sap[75];  
    okup\_aluu.read((char \*) &san, sizeof(double));  
    okup\_aluu.read(sap,20);  
    sap[21]='\0';  
    cout<<san<<' '<<sap;  
    okup\_aluu.close();  
    return 0;  
}

Кийинки програмда файлга *double* тибиндеги массив жазылат дагы кайрадан окулуп, окулган белгилердин санын эсептейт.

#include <iostream>  
#include <fstream>  
#include <cstring>  
using namespace std;  
  
int main()  
{  
    ofstream saktoo("Misal", ios::out | ios:: binary);  
    if(!saktoo)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    double massiv[4]={1.1,1.2,1.3,1.4};  
    saktoo.write((char \*) massiv, sizeof(massiv));  
    saktoo.close();  
    ifstream okup\_aluu("Misal", ios::in | ios:: binary);  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    okup\_aluu.read((char \*) &massiv, sizeof(massiv));  
    int i;  
    for(i=0;i<4;i++)  
        cout<<massiv[i]<<' ';  
    cout<<"\n"<<okup\_aluu.gcount()<<" белги окулду";  
    okup\_aluu.close();  
    return 0;  
}

## **Экилик калыпта киргизүү/чыгаруу. Кошумча маалыматтар**

Мурда көрсөтүлгөн формалардан сырткары *get()* функциясын дагы бир нече жол менен кайрадан жүктөсө болот. Төмөндө эң эле көп колдонгон кайрадан жүктөлүүчү формалардын прототиптери көрсөтүлгөн:

istream &get(char \*буфер, streamsize байттын\_саны);  
istream &get(char \*буфер, streamsize байттын\_саны,char чек);  
int get();

Биринчи функция белгилерди *буфер* көрсөткүчү менен аныкталган массивке окуйт. Окулган белгилердин саны *байттын\_саны* параметри же файлдын айагы белгиси аркылуу аныкталат. *буфер* көрсөткүчү менен аныкталган массивдин айагына *get()* функциясы нөл санын жайгаштырат. Эгерде агымда жаңы сапка өтүү белгиси кезиксе, ал агымдан алынбай, ошол агымда кийинки киргизүү амалына чейин калат.

Экинчи *get()* функциясында окулуучу белгилердин санын *байттын\_саны*параметри же файлдын айагы белгиси же *чек* параметри аркылуу аныкталат. *буфер* көрсөткүчү менен аныкталган массивдин айагына *get()* функциясы нөл санын жайгаштырат. Эгерде киргизүү агымында *чек* белгиси жолукса, ал агымдан алынбай, агымда кийинки киргизүү амалына чейин калат.

Үчүнчү *get()* функциясы агымдан кийинки белгини кайтарып берет. Эгерде файл айагына жетсе, анда ал *EOF* белгисин кайтарат.

Киргизүүнү уйуштуру үчүн *getline()* функциясын колдонсо болот. Бул функция агым аркылуу уйуштурулган киргизүүнүн бардык класстарынын мүчөсү болот. Төмөндө анын прототиптери көрсөтүлгөн:

istream &getline(char \*буфер, streamsize байттын\_саны);  
istream &getline(char \*буфер, streamsize байттын\_саны,char чек);

Биринчи функция белгилерди *буфер* көрсөткүчү менен аныкталган массивке окуйт. Окулган белгилердин саны *байттын\_саны* параметри же файлдын айагы белгиси же жаңы саптын белгиси аркылуу аныкталат. *буфер* көрсөткүчү менен аныкталган массивтин айагына *getline()* функциясы нөл санын жайгаштырат. Эгерде агымда жаңы сапка өтүү белгиси жолукса, ал агымдан алынат, бирок массивке жайгаштырылбайт.

Экинчи *getline()* функциясында окулуучу белгилердин санын *байттын\_саны*параметри же файлдын айагы белгиси же *чек* параметри аркылуу аныктайт. *буфер* көрсөткүчү менен аныкталган массивдин айагына *getline()* функциясы нөл санын жайгаштырат. Эгерде киргизүү агымында *чек* белгиси жолукса, ал агымдан алынат, бирок массивке жайгаштырылбайт.

*getline()* функциясынын эки түрү тең *get()* функциясынын *get(\*буфер, байттын\_саны)*жана *get(\*буфер, байттын\_саны, чек)* түрлөрүнө туура бирдей. Бир гана айырмачылыгы төмөнкүдө: *getline()* функциясы агымдын *чек* белгисин окугандан кийин аны агымдан алып таштайт, ал эми *get()* функциясы ***чек*** белгисин агымдан албайт.

*peek()*функциясы аркылуу агымдан чек белгисин жойбой агымдын кийинки белгисин алса болот. Эгерде файлдын айагы жолукса, анда функция *EOF* маанисин берет. Бул функция агым аркылуу уйуштурулган киргизүүнүн бардык класстарынын мүчөсү болот жана төмөнкү прототипке ээ:

int peek();

*putback()* функциясынын жардамы менен агымдан окулган акыркы белгини кайрадан агымга кайтарса болот. Бул функция агым аркылуу уйуштурулган киргизүүнүн бардык класстарынын мүчөсү болот. Прототиби төмөнкүдөй:

istream &putback(char белги);

мында ***белги*** агымдан окуп алынган акыркы белги.

Чыгаруу амалын аткаруу процессинде берилиштер сырткы эске түздөн-түз жазылбай, убактылуу ички буферде сакталат. Буфер толгон учурда анын ичиндеги берилиштер сырткы эске жазылат. Бирок *flush()* функциясынын жардамы менен буфер толгонго чейин эле берилиштерди сырткы эске жазса болот. Бул функция агым аркылуу уйуштурулган чыгаруунун бардык класстарынын мүчөсү болот жана прототиби төмөнкүдөй:

ostream &flush();

Мисалдарга көңүл буралы. >> оператору аркылуу саптарды окуу убактысында окуп алуу процесси бөлүштүрүү белгиси кезиккенде токтойт. Бирок сапта боштук белгиси бар болсо, андай окуп алуу мүмкүн эмес. *getline()* функциясынын жардамы менен бул маселени чечсе болот.

#include <iostream>  
#include <fstream>  
#include <cstring>  
using namespace std;  
  
int main()  
{  
    char sap[80];  
    cout<<"Атың ким? ";  
    cin.getline(sap,79);  
    cout<<sap<<"\n";  
    return 0;  
}

Эгерде убакыттын ар бир мезгилинде колдонуп жаткан берилиштердин тиби белгисиз болсо, анда *peek()* жана *putback()* функцияларды колдонуу ыңгайлуу болот. Төмөнкү програмда ал ачык-айкын көрүнүп турат. Програмда файлдан саптар же бүтүн сандар окулат. Алар өз ара кандай тартипте жайгашышы мүмкүн.

#include <iostream>  
#include <fstream>  
#include <cctype>  
using namespace std;  
  
int main()  
{  
    char belgi;  
    ofstream saktoo("Misal",ios::out | ios::binary);  
    if(!saktoo)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    char sap[80], \*p;  
    saktoo<<123<<"Bul"<<45;  
    saktoo<<"bizdin"<<67<<"programmabiz"<<endl;  
    saktoo.close();  
     
    ifstream okup\_aluu("Misal",ios::in | ios::binary);  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    do {  
        p=sap;  
        belgi=okup\_aluu.peek();

              //кийинки белгинин тибин аныктоо  
        if(isdigit(belgi)){  
           while(isdigit(\*p=okup\_aluu.get())) p++;

                  //бүтүн санды окуп алуу  
            okup\_aluu.putback(\*p);

                 //белгини агымга кайтаруу  
            \*p='\0';  
            cout<<"Бүтүн: "<<atoi(sap);  
        }  
        else if(isalpha(belgi)){//сапты окуп алуу  
           while(isalpha(\*p=okup\_aluu.get())) p++;  
            okup\_aluu.putback(\*p);

                 //белгини агымга кайтаруу  
            \*p='\0';  
            cout<<"Сап: "<<sap;  
        }  
        else okup\_aluu.get();  
            cout<<'\n';  
    }  
    while (!okup\_aluu.eof());  
        okup\_aluu.close();  
        return 0;  
}

## **Файлга эркин кирүү**

Си++ тилинин киргизүү/чыгаруу тутумунда файлга эркин кирүү *seekg()*жана*seekp()*функциялары аркылуу ишке ашырылат. Алар, ирети менен, киргизүү жана чыгаруу агым функциялары болот. Жалпы түрдө алар төмөнкүчө байандалат:

istream &seekg(off\_type жылышуу, seekdir тапшырма);  
ostream &seekp(off\_type жылышуу, seekdir тапшырма);

Мында *off\_type* параметри *ios* классында аныкталган бүтүн тип. *seekdir* параметри болсо *ios* классында аныкталган аталуучу тип жана ал төмөнкү маанилерге ээ болушу мүмкүн:

|  |  |
| --- | --- |
| **Маани** | **Милдети** |
| ios::beg | Издөөнү файлдын башынан баштоо |
| ios::cur | Издөөнү файлдын учурдагы жеринен  баштоо |
| ios::end | Издөөнү файлдын айагынан баштоо |

Си++ тилинин киргизүү/чыгаруу тутуму файл менен байланышта турган эки көрсөткүчтү башкарат. Биринчиси *окуу көрсөткүчү*. Файлдан ал кийинки окуп алынуучу маалыматтын ордун көрсөтөт. Экинчиси *жазуу көрсөткүчү*. Ал файлга маалымат сакталуучу жердин кийинки ордун көрсөтөт. Ар бир окуу же жазуу амалы аткарылгандан кийин көрсөткүч удаалаш андан ары жылат. Бирок *seekg()*жана *seekp()* функцияларынын жардамы менен файлга ыраатсыз кирсе болот.

*seekg()*функциясы окулуучу көрсөткүчтү *тапшырма* жеринен баштап *жылышуу* өлчөмгө жылдырат. *seekp()*функциясы жазуу көрсөткүчүн *тапшырма* жеринен баштап *жылышуу* өлчөмгө жылдырат.

Эреже бойунча *seekg()*жана *seekp()* функциялары менен амалдарды жүргүзүү үчүн файл экилик код режиминде ачылыш керек.

Ар бир көрсөткүчтүн учурдагы жайгашуу жерин аныктоодо төмөнкү функциялар колдонулат:

pos\_type tellg();  
pos\_type tellp();

Мында *pos\_type* тиби *ios* классында аныкталган бүтүн тип.

Файлдык окуу жана жазуу көрсөткүчтөрүн берилген орунга которуу үчүн *seekg()*жана *seekp()* функцияларынын кайрадан жүктөлүүчү түрүн колдонот. Алардын прототиби төмөнкүдөй:

istream &seekg(pos\_type позиция);  
ostream &seekp(pos\_type позиция);

Төмөнкү програмда *seekp()* функциясы колдонулган. Ал файлдан берилген белгини алмаштырууга жардам берет. Буйрук сапта файлдын ысымын, байттын номерин жана алмаштыруу үчүн белгини колдонуучу киргизет. Файл жазуу жана окуу үчүн ачылат.

#include <iostream>  
#include <fstream>  
#include <cstdlib>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
    if (argc!=4)  
    {cout<<"Алмаштыруу \n";  
        return 1;  
    }  
    fstream saktoo(argv[1], ios::in | ios::out | ios::binary);  
    if(!saktoo)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    saktoo.seekp(atoi(argv[2]),ios::beg);  
    saktoo.put(\*argv[3]);  
    saktoo.close();  
    return 0;  
}

*misal* ысымдуу файлда «*Бул биздин програм*» деген сап бар болсун дейли. Эгерде програмды компилеп буйрук саптан төмөнкү удаалаштыкты терсек:

C:\proba misal 5 $

анда *misal* ысымдуу файлдагы сап төмөнкү сапка алмашат: «*Бул б$здин програм*»

Төмөнкү програмда *seekg()*функциясы окулуучу көрөткүчтү тиешелүү орунга жайгаштырып, ошол орундан баштап калган белгилерди экранга чыгарат. Файлдын ысымын жана окулуп экранга чыгарылуучу биринчи белгинин ордун буйрук сапта берүү зарыл.

#include <iostream>  
#include <fstream>  
#include <cstdlib>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{  
    char belgi;  
    if (argc!=3)  
    {cout<<"Издөө: \n";  
        return 1;  
    }  
    ifstream okup\_aluu(argv[1], ios::in | ios::binary);  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    okup\_aluu.seekg(atoi(argv[2]),ios::beg);  
    while (!okup\_aluu.eof()){  
        okup\_aluu.get(belgi);  
        cout<<belgi;  
    }  
    okup\_aluu.close();  
    return 0;  
}

## **Киргизүү/чыгаруу абалын текшерүү**

Си++тилинде ар бир киргизүү/чыгаруу амалы аткарылгандан кийин, ал амалдын абалы киргизүү/чыгаруу тутумунда сакталат. Андай маалымат *ios* классында аныкталган *iostate* тибиндеги объекте сакталат. Ал төмөнкү мүчөлөргө ээ:

|  |  |
| --- | --- |
| **Маани** | **Жыйынтык** |
| goodbit | Ката жок |
| eofbit | Файлдын айагы |
| failbit | Фаталдуулук эмес ката |
| badbit | Фаталдуу ката |

Киргизүү/чыгаруу абалын билүүнүн эки жолу бар. Биринчиси, *ios* классынын мүчөсү болгон *rdstate()* функциясын чакыруу аркылуу. Бул функция ката байрактар абалын кайтарып берет. Эч кандай ката жок болсо, бул функция *goodbit* маанисин, калган учурда тиешелүү катанын байрагын берет. Анын прототиби төмөнкүдөй:

iostate rdstate();

Экинчи жол катары *ios* классынын мүчөсү болгон төмөнкү бир нече функцияны колдонуу аркылуу аныктаса болот:

bool bad();  
bool eof();  
bool fail();  
bool good();

Эгерде туу *badbit* абалында болсо, анда *bad* функциясы чын деген маанини кайтарып берет. *eof()* функциясын жогоруда талкулаганбыз. *fail()* функциясы чын деген маанини берет, эгерде *failbit* байрагы орнотулса. Ката жок болгон учурда *good()* функциясы чын деген маанини берет. Калган учурларда бул функциялар жалган деген маани кайтарат.

Ката пайда болгон учурда ал каталарды эсепке албай, програмдын аткарылышын улантууга туура келсе, *ios* классынын мүчөсү болгон *clear()* функциясын колдонот. Анын прототиби төмөнкүдөй:

void clear(iostate байрактар=ios::goodbit);

Эгерде *байрактар* параметри *goodbit* барабар болсо, анда бардык каталардын байрактары эсепке алынбайт. Калган учурларда *байрактар* өзгөрмөсүнө эсепке алынбаган байрактардын маанилери менчиктелет.

Төмөнкү програмда *rdstate()* функциясын аткарылышы сүрөттөлгөн. Програм экранга тексттик файлдын мазмунун чыгарат. Ката пайда болгон учурда ката жөнүндөгү маалымат *abal* функциясы аркылуу маалымдалат.

#include <iostream>  
#include <fstream>  
using namespace std;  
void abal(ifstream &in);  
  
int main(int argc, char \*argv[])  
{  
    if (argc!=2)  
    {cout<<"Файлдын мазмуну:\n";  
        return 1;  
    }  
    ifstream okup\_aluu(argv[1]);  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    char belgi;  
    while (okup\_aluu.get(belgi))  
    {cout<<belgi;  
        abal(okup\_aluu);  
    }  
    abal(okup\_aluu);  
    okup\_aluu.close();  
    return 0;  
}  
  
void abal(ifstream &in)  
{  
    ios::iostate i;  
    i=in.rdstate();  
    if(i & ios::eofbit)  
        cout<<"Файлдын айагы\n";  
    else  
    if(i & ios::failbit)  
        cout<<"Киргизүү/чыгаруунун фаталдуу эмес катасы\n";  
    else  
    if(i & ios::badbit)  
        cout<<"Киргизүү/чыгаруунун фаталдуу катасы\n";  
}

Бул програм жок дегенде «Файлдын айагы» деген маалыматты берет.

Төмөнкү програмда *good()* функциясынын жардамы менен файлда ката бар же жок экендиги аныкталат.

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main(int argc, char \*argv[])  
{ char belgi;  
    if (argc!=2)  
    {cout<<"Файлдын мазмуну:\n";  
        return 1;  
    }  
    ifstream okup\_aluu(argv[1]);  
    if(!okup\_aluu)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    while (!okup\_aluu.eof())  
    {okup\_aluu.get(belgi);  
        if(!okup\_aluu.good() && !okup\_aluu.eof())  
        {cout<<"Киргизүү/чыгаруу катасы\n";  
        return 1;  
        }  
    cout<<belgi;  
        }  
        okup\_aluu.close();  
    }

## **Киргизүү/чыгарууну кайрадан жүктөө жана файлдар**

Алдыда түзүлүп жаткан класс үчүн киргизүү/чыгаруу операторлорунун кайрадан жүктөө ыкмаларын карадык. Си++тилинде бардык агымдар бирдей болгондуктан, экранга чыгаруу үчүн кайрадан жүктөлгөн  киргизүү/чыгаруу операторлорун файлга киргизүү/чыгаруу үчүн колдонуу каралган.

Кайрадан жүктөлгөн киргизүү/чыгаруу оператор жана манипуляторлорду каалаган агым менен колдонууга болот. Бирок, эгерде киргизүү/чыгаруу функцияларын айкын бир агымга байласак, анда киргизүү/чыгаруу функцияларынын колдонуу аймагы ошол агым менен чектелип калат. Ошондуктан мындай функцияларды иштеп чыгууда алар каалагандай агым менен иштөөгө мүмкүнчүлүгү болгондой иштеп чыгуу керек.

Төмөнкү програмда *misal* классына карата **<<** жана **>>** операторлору кайра жүктөлөт. Оператор-функциялар менен экранга дагы, файлга дагы чыгарууга болот.

#include <iostream>  
#include <fstream>  
using namespace std;  
  
class misal{  
    int i,j;  
    public:  
        misal(int x, int y){i=x;j=y;}  
        friend ostream &operator<<(ostream &agym, misal object);  
        friend istream &operator>>(istream &agym, misal &object);  
    };  
  
    ostream &operator<<(ostream &agym, misal object)  
    {  
        agym<<object.i<<' '<<object.j<<"\n";  
        return agym;  
    }  
  
    istream &operator>>(istream &agym, misal &object)  
    {  
        agym>>object.i>>object.j;  
        return agym;  
    }  
  
    int main()  
{  
    misal object\_1(1,2), object\_2(3,4);  
    ofstream saktoo("Proba");  
    if(!saktoo)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    saktoo<<object\_1<<object\_2;  
    saktoo.close();  
    ifstream okup\_aluu("Proba");  
    if(!okup\_aluu)  
        {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    misal object\_3(0,0), object\_4(0,0);  
    okup\_aluu>>object\_3>>object\_4;  
    cout<<object\_3<<' '<<object\_4;  
    okup\_aluu.close();  
    return 0;  
}

Киргизүү/чыгаруу манипуляторлорунун бардыгын файлга киргизүү/чыгарууда колдонууга болот. Төмөнкү програмда маалыматты экранга чыгарган манипуляторду файлга жазуу үчүн колдонулду.

#include <iostream>  
#include <fstream>  
#include <iomanip>  
using namespace std;  
  
ostream &manip\_1(ostream &agym)  
{agym<<"Көңүл бургула: ";  
    return agym;  
}  
  
ostream &manip\_2(ostream &agym)  
{agym<<"унутпагыла!!!";  
    return agym;  
}  
  
int main()  
{ofstream saktoo("Proba");  
    if(!saktoo)  
    {cout<<"Файлды ачууга мүмкүн эмес\n";  
        return 1;  
    }  
    //Экранга чыгаруу  
    cout<<manip\_1<<"шыкалган ток!!!\n";  
    cout<<"Жарыкты өчүргөндү "<<manip\_2;  
    //Файлга сактоо  
    saktoo<<manip\_1<<"шыкалган ток!!!\n";  
    saktoo<<"Жарыкты өчүргөндү "<<manip\_2;  
    saktoo.close();  
    return 0;  
}

## **КӨНҮГҮҮЛӨР**

1.       Файлда жайгашкан калкыган чекиттүү сандарды окуп, окулган түгөйлөрдөн комплекс сандарын түзгүлө. Түзүлгөн сандарды экранга чыгарып аларды жаңы файлга сактагыла

2.       Кандайдыр бир сапты киргизүүнү талап кылган класс үчүн << амалын кайрадан жүктөгүлө. Класс киргизилген сапты экранга чоң тамгаларды жана цифраларды жок кылып чыгарыш керек.

3.       Ар кандай түрдөгү маалыматты суроо-талап берген жана окуган бир нече функцияларды түзгүлө (жөнөкөй мисал Ооба/Жок суроо-талапты аткарган функция) Идея: почто дареги, туулган жылы, өздүк маалыматтар ж.б. Туура эмес киргизүүдөн корголгондой болсун.

4.       Киргизилген саптан төмөнкүлөрдү чыгаргыла: (1) кичинекей тамагларды, (2) бардык тамгаларды, (3) бүт тамга жана цифраларды, (4) чоң тамгаларды, (5) тыныш белгилерди.

5.       Си++ тилинин стандарттуу китепканасынын жардамы менен Си тилинин киргизүү/чыгаруу функцияларын ишке ашыргыла (printf и scanf).

6.       Файлдан символдорду окуп алуу үчүн [] операторун кайрадан жүктөгөн классты ишке ашыргыла.

7.       Төмөнкү манипуляторлорду түзгүлө: 1) символдорду чоң тамгалар менен ал эми сандарды OCT калыпта чыгарган; 2) тамгаларды кичинекей тамгалар менен жана боштуктарды # белгиси менен алмаштырган; 3) сандарды HEX калыпта чыгарып боштуктарды чыгарбаган.

8.       Символдордун сабын талап кылган жана ал символдорду экранга төмөнкү калыптарда чыгарган програм түзгүлө: 1) тамгалардын бардыгы төмөнкү регистрде; 2) тамгалардын барлыгын жогорку регистрде;3) сандарды HEX калыпта чыгаруу; 4) боштуктарды # белгисине алмаштыруу.

9.       cout жардамы менен printf функциясын, scanf жардамы менен cin операторлорун бир мезгилде колдонууга мүмкүнчүлүк берген програмды ишке ашыргыла.

10.    cin жардамы менен scanf функциясын, printf жардамы менен cout операторлорун бир мезгилде колдонууга мүмкүнчүлүк берген програмды ишке ашыргыла.

11.    Сандарды бир тутумдан башка тутумга которгон програмды түзгүлө (HEX,BIN,OCT).

12.    Бирден миңге чейин 256 кокус сандарды камтыган массивди түзгөн програмды түзгүлө. Элементтерди төмөнкү иретте экранга чыгаргыла: биринчи DEC калыпта, экинчи HEX калыпта, үчүнчү OCT калыпта, төртүнчү кайрадан DEC калыпта ж.б.у.с.

13.    Инициалоодо (конструктордо) файлдын ысымын алган жана деструктордо алгачкы файлдагы тексттик саптан бардык боштуктарды жойуп, бөлчөк сандардын бөлчөк бөлүмүн 2 цифрага чейин кыскарткан жана кичинекей тамгаларды чоң тамгаларга алмаштырган классты түзгүлө.

14.    Комплекс сандарды сактаган файлды окуп, ар бир түгөйдөн жөнөкөй сан түзүп жаңы файлга сактагыла.

15.    Каалаган сапты киргизүүнү талап кылган жана ал сапты экранга кичинекей тамаглары жок кылып чыгарган класс үчүн << амалын кайрадан жүктөгүлө.

16.    Саптагы тамгалардын бардыгын чоң тамгалар менен, ал эми сандарды HEX калыпта чыгарган класс үчүн >> амалын кайрадан жүктөгүлө.