# **Ысымдар мейкиндиги жана кошумча маалыматтар**

## **Ысымдар мейкиндиги**

Си++тилинде ысымдар мейкиндиги ысымдардын өз карама-каршы кырдаалдарын жойуу максатында колдонулат. Си++тилинде көп сандагы өзгөрмөлөрдүн, функциялардын жана класстардын ысымдары пайдаланылат. Ысымдар мейкиндиги пайда болгонго чейин бул ысымдар бир глобалдуу ысымдар мейкидигинде жайгашчу. Ошондуктан ар кандай карама-каршылыктар көп пайда болчу. Мисалы, програмда *tolower()* функциясын аныктасак, ал функция китепканада жайгашкан *tolower()* милдетин аткарып калчу. Програмда ар кандай өндүрүүчүлөр тарабынан иштелип чыккан функция жана класстарды колдонуу учурунда да карама-каршылыктарга алып келчү.

Бул маселе ысымдар мейкиндиги түшүнүгү жана *namespace*кызматчы сөзү киргизилгенден кийин чечилди. Кызматчы сөз аркылуу берилген ысымдар мейкиндигинде ысымдардын көрүү аймагын чектөөгө мүмкүндүк ачылды. Ысымдар мейкиндиги бир ысымды ар кандай контекстте колдонууга уруксат берет. Си++ тилинин китепканасы өзүнүн *std* мейкиндигинде аныкталат. Бирок програмда менчик ысымдар мейкиндиктерин түзсө болот.

Ысымдуу аймактарды кошуу менен *namespace*кызматчы сөзү глобалдуу ысымдар мейкиндигин ажыратууга мүмкүндүк берет. Чындыгында ысымдар мейкиндиги көрүү аймагын аныктайт. Төмөндө *namespace*кызматчы сөзүн колдонуунун синтаксиси келтирилген:

namespace ысым{  
//жарыялоолор  
}

*namespace*көрсөтмөнүн ичинде аныкталган жарыялоолор берилген ысымдар мейкиндиктин көрүү аймагынын ичинде жатат. Мисалы,

namespace meykindik{  
int i,j;  
void func(int k){cout<<k;}  
  
class misal{  
public:  
void tanda(int x){i=x;}  
int kaytar(){return i;}  
};  
}

Мында*i, j*өзгөрмөлөрүнүн, *func()*функциянын жана *misal*классынын ысымдары *meykindik* ысымдар мейкиндиги менен аныкталган көрүү аймагында жатат.

Ысымдар мейкиндигинде жарыяланган идентификаторлорго ал мейкиндик ичинен түздөн-түз кайрылса болот. Мисалы, *kaytar()* функциясындагы *return i;* көрсөтмөсүндө *i*  өзгөрмөсү ачык көрсөтүлгөн. Ысымдар мейкиндигинде аныкталган объектерге ал мейкиндиктин сыртынан кайрылганда көрүү аймагын кеңейтүү операторун колдонуш керек. Мисалы, *i* өзгөрмөсүнө 5 деген маанини менчиктөө үчүн төмөнкү көрсөтмөнү колдонуш керек:

meykindik::i=5;

Ал эми *misal* тибиндеги объектини жарыялоо үчүн төмөнкү көрсөтмө керек:

meykindik:: misal object;

Эгерде програмдагы ысымдар мейкиндигинин мүчөлөрүнө өтө көп кайрылуу жүргүзүлсө, анда кайрылуу процессин жеңилдетүү үчүн *using* көрсөтмөсү колдонот. Бул көрсөтмөнүн эки калыбы бар:

using namespace ысым;  
using ысым::мүчө;

Биринчи калыпта *ысым* параметри кирүүгө мүмкүндүк алып жаткан ысымдар мейкиндигин көрсөтөт. *using* көрсөтмөнүн бул калыбын колдонууда аталган ысымдар мейкиндигинде аныкталган бардык мүчөлөрүнө түздөн-түз кайрылууга болот. *using* көрсөтмөнүн экинчи калыбын колдонууда көрсөтмөдө көрсөтүлгөн мүчө гана ачык болот. Жогоруда келтирилген ысымдар мейкиндигине карата төмөнкү менчиктөө операторлор туура:

using meykindik::j;  
j=10;  
using namespace meykindik;  
i=15;

Бир ысымга ээ болгон бир нече ысымдар мейкиндигин жарыялоо каралган. Мындай ыкма ысымдар мейкиндигин бир нече файлга же бир файлдын ичинде бир нече ысымдар мейкиндигин аныктоого мүмкүндүк берет. Төмөнкү мисалга назар салалы:

namespace meykin{  
int i;  
}  
//…  
namespace meykin{  
int j;  
}

Мында *meykin* ысымдар мейкиндиги эки бөлүккө бөлүнгөн. Бирок ар бир бөлүктүн мазмуну бир *meykin* ысымдар мейкиндигинде калат.

Ысымдар мейкиндигинин өзгөчө тиби бар – *ысымсыз ысымдар мейкиндиги*. Ысымсыз мейкиндиктер бир файлдын ичинде уникалдуу болгон идентификаторлорду түзүүгө жардам берет. Файлдын ичинен андай идентификаторлорго түздөн-түз кайрылууга болот. Файлдын сыртынан ал идентификаторлорго кайрылууга мүмкүн эмес. Төмөндө ысымсыз мейкиндиктин негизги калыбы келтирилген:

namespace{  
//жарыялоолор  
}

Төмөнкү програмда ысымдар мейкиндигинин ажыралгыс касиеттери көрсөтүлгөн.

#include <iostream>  
using namespace std;  
  
//Биринчи ысымдар мейкиндигин аныктоо  
namespace bir\_im{  
  
class misal{  
    int i;  
    public:  
        misal(int x){i=x;}  
        void al(int x){i=x;}  
        int ber(){return i;}  
    };  
  
    char sap[]="Ысымдар мейкидигине мисал.\n";  
    int k;  
}  
  
//Экинчи ысымдар мейкиндигин аныктоо  
namespace eki\_im{  
    int x,y;  
}  
  
int main()  
{  
//Көрүү аймагын кеңейтебиз     
bir\_im::misal object(5);  
cout<<"Объектинин мааниси ="<<object.ber()<<endl;  
object.al(10);  
cout<<"Эми объектинин мааниси ="<<object.ber()<<endl;  
//sap сабын учурдагы коруу аймагына киргизебиз  
using bir\_im::sap;  
cout<<sap;  
//bir\_im ысымдар мейкиндигин толугу менен  
//көрүү аймагына киргизебиз  
using namespace bir\_im;  
for(k=1;k<=5;k++)  
    cout<<k<<' ';  
cout<<endl;  
//eki\_im ысымдар мейкиндигин колдонобуз  
eki\_im::x=10;  
eki\_im::y=20;  
cout<<"x="<<eki\_im::x<<endl;  
cout<<"y="<<eki\_im::y<<endl;  
//eki\_im ысымдар мейкиндигин толугу менен  
//көрүү аймагына киргизебиз  
using namespace eki\_im;  
misal object\_1(x), object\_2(y);  
cout<<"Объект\_1="<<object\_1.ber()<<endl;  
cout<<"Объект\_2="<<object\_2.ber()<<endl;  
return 0;  
}

Програм аткарылганда экранга төмөнкү жыйынтык чыгарылат:

Объектинин мааниси =5  
Эми объектинин мааниси =10  
Ысымдар мейкидигине мисал.  
1 2 3 4 5  
x=10  
y=20  
Объект\_1=10  
Объект\_2=20

Програмда бир нече ысымдар мейкиндигин колдонулганда бир ысымдар мейкиндиги башка ысымдар мейкиндигин алмаштырбайт. Учурдагы көрүү аймагына жаңы ысымдар мейкиндиги киргизилгенде, ал мейкиндик көрүү аймагына толукталат. Биздин програм аткарылып бүткөндөн кийин ысымдардын глобалдуу мейкиндиги *std, bir\_im*жана *eki\_im* ысымдар мейкиндиктери менен толукталып калат.

Ысымдар мейкиндигин файлдардын арасына же бир файлдын ичинде ажыратса болот. Мындай учурларда андай ысымдар мейкиндигинин мазмундары бириктирилет. Мисалы,

#include <iostream>  
using namespace std;  
  
//Ысымдар мейкиндигинде i өзгөрмөсүн аныктоо  
namespace meykindik{  
    int i;  
}  
  
//Ысымдар мейкиндигинде j өзгөрмөсүн аныктоо  
namespace meykindik{  
    int j;  
}  
  
//Өзгөрмөнү ысымдардын глобалдуу мейкиндигинде аныктоо  
int a;  
  
int main()  
{  
using namespace meykindik;  
    i=5;  
    j=10;  
    a=15;  
    cout<<i<<' '<<j<<' '<<a;  
return 0;  
}

Бул мисалда *i* жана *j* өзгөрмөлөрү *meykindik* ысымдар мейкиндигинде жайгашты, ал эми *а* өзгөрмөсү ысымдардын глобалдуу мейкиндигинде аныкталды.

Си++ тилинин стандарты *std* ысымдар мейкиндигинде атайын китепкананы аныктайт. Ошондуктан програмдын башына төмөнкү көрсөтмө жазылат:

using namespace std;

Бул көрсөтмө *std* ысымдар мейкиндигин учурдагы ысымдар мейкиндигине айландырат дагы, Си++ тилинин стандартында аныкталган класстар жана функциялардын ысымдарына *std* ысымдар мейкиндигин атабастан, түздөн-түз кайрылууга мүмкүндүк берет. Ошого карабай ар бир идентификатордун алдына *std* ысымдар мейкиндигин атаса болот. Төмөнкү програмда *std* ысымдар мейкиндиги глобалдуу көрүү аймагына киргизилген эмес.

#include <iostream>  
int main()  
{int i;  
    std::cout<<"Бир сан киргиз: ";  
    std::cin>>i;  
    std::cout<<"Киргизилген сан =";  
    std::cout<<i;  
return 0;  
}

Стандарттуу киргизүү/чыгаруу агымдардын алдында ысымдар мейкиндиги айкын көрсөтүлгөн.

Эгерде програмда стандарттуу китепканадан бир нече гана ысым колдонсо, анда *using* көрсөтмөсүн колдонуп, ал ысымдарды атап чыгуу ыңгайлуу болот. Төмөнкү мисалды карап көрөлү:

#include <iostream>  
using std::cout;  
using std::cin;  
  
int main()  
{int i;  
    cout<<"Бир сан киргиз: ";  
    cin>>i;  
    cout<<"Киргизилген сан =";  
    cout<<i;  
return 0;  
}

## **Кайрадан өзгөртүү функциялары**

Кээ бир учурда бир типтеги объекти башка типтеги объектке өзгөртүүгө туура келет. Мындай өзгөртүүнү өзгөртүү функция аркылуу ишке ашырылат. Өзгөртүү функция объекти башка тип менен сиңишүүчү же Си++ тилинде аныкталган стандарттуу типке айландырат. Төмөндө өзгөртүү функциянын негизги калыбы келтирилген:

operator тип(){return маани;}

мында ***тип*** өзгөртүүнүн максаттуу тиби, ***маани*** объектин өзгөртүлгөн мааниси. Өзгөртүү функциясы ***тип*** тибиндеги маанини кайтарып берет. Өзгөртүү функция параметрге ээ болбойт жана ал өзгөртүү жүргүзүлүп жаткан класстын мүчөсү болуш керек.

Төмөнкү програмда *misal* класс объекти бүтүн типке айландырган өзгөртүү функциясын камтыйт. Функция эки сандын көбөйтүндүсүн кайтарып берет.

#include <iostream>  
using namespace std;  
  
class misal{  
    int i,j;  
    public:  
        misal(int x, int y){i=x;j=y;}  
        operator int(){return i\*j;}//Өзгөртүү функциясы  
    };  
  
    int main()  
    {  
        misal object\_1(1,2), object\_2(3,4);  
        int a;  
        a=object\_1;//Объект бүтүнгө айланат  
        cout<<a<<"\n";  
        a=10+object\_2;//Объект бүтүнгө айланат  
        cout<<a;  
        return 0;  
}

Жыйынтык катары 2 жана 22 маанилери чыгат.

Төмөнкү мисалда *misal*тибиндеги сап символдук шилтемеге өзгөртүлөт.

#include <iostream>  
#include <cstring>  
using namespace std;  
  
class misal{  
    char sap\_1[80];  
    int k;  
    public:  
        misal(char \*s){strcpy(sap\_1,s); k=strlen(s);}  
        operator char\*(){return sap\_1;}//өзгөртүү функциясы  
    };  
  
    int main()  
    {  
        misal s("Салам\n");  
        char \*p, sap\_2[80];  
        p=s;//char \* тибине айландыруу  
        cout<<"Бул сап: "<<p<<"\n";  
        strcpy(sap\_2,s);  
        cout<<"Бул саптын көчүрмөсү: "<<sap\_2;  
        return 0;  
}

Програм экранга төмөнкү жыйынтыкты чыгарат:

Бул сап: Салам  
Бул саптын көчүрмөсү: Салам

Өзгөртүү функциясы *s* объекти *р* объектке менчиктелгенде жана *strcpy()* функциясын колдонгондо чакырылат.

## **Класстын статик мүчөлөрү**

Класстын мүчөсү болгон өзгөрмөлөрдү статик (*static*) өзгөрмө катары жарыяласа болот. Статик мүчө-өзгөрмөлөрдү колдонуп бир катар маселелер чечилет. Эгерде өзгөрмө статик өзгөрмө катары жарыяланса, анда берилген класстагы объектердин санына карабай, статик өзгөрмөнүн бир гана көчүрмөсү түзүлөт. Ар бир объект башка объектер менен бирдикте бир гана өзгөрмөнү колдонот. Андан сырткары, статик өзгөрмөнү класстын туунду класстарында колдонууга болот. Эске салсак, жөнөкөй өзгөрмө-мүчөлөрдү колдонууда ар бир объектке өзүнчө көчүрмө түзүлөт жана көчүрмөгө кайрылуу ошол объект аркылуу аткарылат.

Демек класстын статик мүчөсү жөн эле глобалдуу өзгөрмө. Анын көрүү аймагы ал өзгөрмө жарыяланган класс менен чектелет. Ошондуктан статик өзгөрмөгө эч кандай объект менен байланыш түзбөй, түздөн-түз кайрылса болот.

Класстын ичинде статик берилиш-мүчөлөрдү жарыялаганда, биз аларды аныктабайбыз. Аныктоону класстын сыртында жүргүзүү керек. Статик өзгөрмө кайсы класска таандык экендигин көрсөтүү үчүн, анын көрүү аймагын кеңейтүү амалын колдонуп, кайрадан жарыялаш керек (*redeclare*).

Бардык статик мүчө-өзгөрмөлөргө унчукпаганда баштапкы маани катары нөл берилет. Бирок, зарыл болгон учурда, класстын статик мүчөсүнө каалгандай баштапкы маани берүүгө тыйуу салынбайт..

Төмөндө статик мүчө-өзгөрмөнү колдонууга жөнөкөй мисал келтирилген.

#include <iostream>  
using namespace std;  
class misal{  
    static int i;  
    public:  
        void tanda(int x){i=x;}  
        int kaytar(){return i;}  
    };  
  
//misal::i аныктоо. i өзгөрмөсү misal классынын  
//жабык мүчөсү  
int misal::i;  
  
    int main()  
    {  
        misal object\_1, object\_2;  
        object\_1.tanda(5);  
        cout<<"Объект\_1="<<object\_1.kaytar()<<"\n";  
        cout<<"Объект\_2="<<object\_2.kaytar()<<"\n";  
return 0;  
}

Бул програм төмөнкү жыйынтыкты берет:

Объект\_1=5  
Объект\_2=5

Мында *object\_1* объекти үчүн статик мүчө-өзгөрмөнүн мааниси менчиктелет. Бирок, ал өзгөрмө *object\_1* жана *object\_1* бирге колдонгон үчүн *kaytar()* функциясы бирдей жыйынтык берди? *i* өзгөрмөсү *misal* классынын ичинде жарыяланып класстын сыртында аныкталды. Жарыялоо учурунда өзгөрмөгө эс бөлүнбөйт. Статик өзгөрмөгө эс бөлүү үчүн, ага маани берип, айкын аныкташ керек.

Класстын мүчөсү болгон статик өзгөрмө объект түзүлгөнгө чейин жарыяланган үчүн, ал өзгөрмөгө объекти аркылуу эмес, түздөн-түз кайрылса болот. Төмөнкү програмда статик өзгөрмөгө маани объектиге шилтеме берилбей аныкталып жатат.

#include <iostream>  
using namespace std;  
class misal{  
    public:  
        static int i;  
        void tanda(int x){i=x;}  
        int kaytar(){return i;}  
    };  
int misal::i;  
    int main()  
    {  
        misal object\_1, object\_2;  
        //Өзгөрмөгө түздөн-түз маани берүү  
        misal::i=5;  
        cout<<"Объект\_1="<<object\_1.kaytar()<<"\n";  
        cout<<"Объект\_2="<<object\_2.kaytar()<<"\n";  
return 0;  
}

Статик өзгөрмөгө 5 деген маани берилгендиктен, төмөнкү жыйынтыкты алабыз:

Объект\_1=5  
Объект\_2=5

Класста статик мүчө-функцияларды жарыяласа болот. Класстын жарыяланган мүчө-функциясына ал класстын башка статик мүчөлөрү аркылуу гана кайрылууга болот. Статик мүчө-функцияларында *this* көрсөткүчү жок. Алар виртуал боло албайт. Статик мүчө-функцияларды *const* (турактуу) жана *volatile* (өзгөрмөлүү) идентификаторлор менен жарыялоого болбойт. Класстын статик мүчө-функциялары ал класстын каалаган объекти аркылуу чакырылат. Ошондой эле класстын статик мүчө-функциясын класстын бир дагы объектине байланыштырбай, класстын ысымы жана көрүү аймакты кеңейтүү оператору аркылуу чакырылат.

Статик мүчө-функциялар сейрек колдонот, бирок реалдуу объект түзүлүүдө жабык статик мүчөлөрдү алдын-ала аныктоо үчүн колдонуу ыңгайлуу болот.

Мисалы,

#include <iostream>  
using namespace std;  
class misal{  
       static int i;  
       public:  
       static void tanda(int x){i=x;}  
       void korsot(){cout<< i;}  
    };  
    int misal::i;  
    int main()  
    {  
        misal::tanda(5);  
        misal x;  
        x.korsot();  
    return 0;  
}

Бул програмда *tanda()*функциясынын мааниси *i* өзгөрмөсүн *misal*тибиндеги объект түзүлгөнгө чейин алдын-ала аныктайт.

## **Класстын турактуу жана өзгөрмө мүчөлөрү**

Класстын мүчө-функциялары *const* идентификатордун жардамы менен турактуу деп жарыяланышы мүмкүн. Эгерде функция турактуу деп жарыяланса, анда ал аны чакырган объекти өзгөртө албайт. Андан сырткары турактуу объект турактуу эмес мүчө-функцияны чакыра албайт, бирок турактуу мүчө-функция турактуу жана турактуу эмес объектерге кайрыла алат.

Турактуу мүчө-функцияны түзүү үчүн төмөнкү мисалда келтирилген калып колдонулат:

class misal{  
int ozgormolor;  
public:  
int funksiya() const;//турактуу мүчө-функция

};

*const* кызматчы сөзү функциянын ысымынын алдында эмес, параметрлердин тизмесинен кийин турат. Мисалы,

#include <iostream>  
using namespace std;  
class misal{  
       int i;  
       public:  
       int fun\_1() const {return i;}  
       void fun\_2(int x) const{i=x;}  
    };  
    int main()  
    {misal object;  
object.fun\_2(5);  
cout<< object.fun\_1();  
return 0;

}

Компилөө учурунда бул програм ката жөнүндө маалымат берет, себеби *fun\_2()* турактуу функция аркылуу*i* өзгөрмөсүнүн маанисин өзгөртүүгө аракет жасалып жатат. Бирок *fun\_1()* функциясы туура, себеби ал функция *i* өзгөрмөсүнүн маанисин өзгөртпөйт.

Турактуу мүчө-функция аркылуу класстын мүчөлөрүн өзгөртсө болот. Ал үчүн өзгөртүүлүчү мүчөлөрдү *mutable* кызматчы сөзү менен байандайбыз. Мисалы,

#include <iostream>  
using namespace std;  
class misal{  
       mutable int i;  
    int j;  
       public:  
       int fun\_1() const {return i;}  
       void fun\_2(int x) const{i=x;}  
       /\*void fun\_3(int x) const{j=x;} //Бул сап ката  
        };

    int main()  
    {misal object;  
     object.fun\_2(5);  
     cout<< object.fun\_1();  
return 0;

}

Програмда *i*өзгөрмөсү өзгөртүүлүчү өзгөрмө катары байандалды, ошондуктан анын маанисин өзгөрттүк. Ал эми *j*өзгөрмөсүнүн маанисин*fun\_3()*фукнциясы аркылуу өзгөртө албайбыз.