**Мурастоого киришүү**

Си++тилинде мурастоо бир класс башка класстын касиеттерин мурастоо механизмин ишке ашырган каражат болот. Мурастоо аркылуу жалпы касиеттен өзгөчө касиеттерге өтүү жолу аркылуу класстар иерархиясын түзсө болот.

Мурастоону талкулоодо төмөнкү эки терминди аныктайлы. Бир класс башка класстан мурас алганда мурас берген классты ***база класс,*** ал эми мурас алган классты ***туунду класс***деп аташат. Мурас алуу процесси база классты түзүүдөн башталат. База класс туунду класстар үчүн жалпы болгон сапаттарды аныктайт, б.а. база класс мүнөзү жалпы болгон касиеттерди аныктайт. Туунду класс болсо ал жалпы касиеттерди мурастайт дагы, өзүнө мүнөздүү болгон касиеттерди кошот.

Бир класс башка класстан мурасты кандайча алаарын байкоо үчүн жөнөкөй мисалга көңүл буралы:

//база классы  
class A{  
int i;  
public:  
void tandoo\_i(int n);  
int aluu\_i();  
};  
  
//База класстан мурас алган туунду класс  
class B:public A{  
int j;  
void tandoo\_j(int n);  
int koboyt();  
};

Жарыялоодо *В* классынын ысымынан кийин кош чекит, андан кийин *public* кызматчы сөзү жана *А* классынын ысымы жазылган. Компилегич *В* классы *А* классынын бардык компонентерин мурастайт деген көрсөткүчтү кабыл алат. *public* кызматчы сөзү компилегичге *А* классынын баардык ачык мүчөлөрү *В* классы үчүн ачык болоорун билдирет. Бирок база класстын жабык мүчөлөрү жабык болот жана аларга туунду класстан түз кайрылууга болбойт. Жогорууда келтирилген програмдын үзүндүсүнүн толугу менен ишке ашырылышы төмөндө берилди.

#include <iostream>  
using namespace std;  
  
//база классты аныктоо  
class A{  
int i;  
public:  
void tandoo\_i(int n);  
int aluu\_i();  
};  
  
// Туунду классты аныктоо  
class B:public A{  
int j;  
    public:  
void tandoo\_j(int n);  
int summa();  
};  
  
//База класста i өзгөрмөсүн аныктоо  
void A::tandoo\_i(int n)  
{i=n;}  
int A::aluu\_i()  
{return i;}  
  
//Туунду класста j өзгөрмөсүн аныктоо  
void B::tandoo\_j(int n)  
{j=n;}  
/\*База класстын i жана туунду класстын j маанилерин кайтарып берүү\*/  
int B::summa()  
{  
/\*туунду класстан база класстын мүчө-функцияларын чакырса болот\*/  
    return j+aluu\_i();  
}  
  
int main()  
{  
B object;  
object.tandoo\_i(3); //База класска жүктөө  
object.tandoo\_j(5);// Туунду класска жүктөө  
cout<<object.summa();// 15 санын чыгаруу  
return 0;  
}

*summa()*функциясынын аныктоосунан назар алсак, база класстын мүчөсү болгон  *aluu\_i()*функциясы *В*классынан эч кандай объектке байланбай чакырылып жатат. Мындай абал база класстын ачык мүчөлөрү туунду класс үчүн ачык болгондуктан келип чыкты. *summa()*функциясында *i* өзгөрмөсүнө биз *aluu\_i()* функциясы аркылуу кайрылдык, себеби база класстын жабык мүчөлөрүнө туунду класстан кайрылууга болбойт.

**База класска кирүү мүмкүндүгүн башкаруу**

Бир класс башка класстан мурас алганда төмөнкү жалпы калып колдонулат:

сlass туунду\_класстын\_ысымы:жетүү\_ыкмасы база\_класстын\_ысымы  
{  
//…  
}

Мында *жетүү\_ыкмасы* төмөнкү үч кызматчы сөздүн бири болушу мүмкүн: *public, private*же *protected****.*** Бул кызматчы сөздөрдү *жетүү спецификаторлору* деп аташат. Жетүү спецификаторлору база класстын элементтерин туунду класс кандайча мурас аларын билдирет. Эгерде база класстын жетүү спецификатору *public* деп жарыяланса, анда база класстын бардык ачык мүчөлөрү туунду класс үчүн да ачык болот. Эгерде база класстын жетүү спецификатору *private* деп жарыяланса, анда база класстын бардык ачык мүчөлөрү туунду класс үчүн жабык болот. Ал эми *рrotected* спецификатору *private* спецификаторун окшош. Бир гана айырмасы бар: база класстын коргологон мүчөлөрү туунду класстар үчүн ачык болот, бирок база жана туунду класстардын сыртында ал мүчөлөргө кайрылууга болбойт.

Төмөнкү мисалда база жана туунду класстар келтирилген. Мурастоо *public* спецификатору менен жүргүзүлөт:

#include <iostream>  
using namespace std;  
//база классты аныктоо  
  
class basa{  
int i;  
public:  
void tanda\_i(int x){i=x;}  
void korsot\_i() {cout<<i<<"\n";}  
};  
  
//Класс ачык класс катары мурасталат  
class tuundu:public basa{  
int j;  
public:  
void tanda\_j(int y){j=y;}  
void korsot\_j() {cout<<j<<"\n";}  
};  
  
int main()  
{  
tuundu object;  
object.tanda\_i(3); //База класстын мүчөсүнө кайрылуу  
object.tanda\_j(5);// Туунду класстын мүчөсүнө кайрылуу  
object.korsot\_i();//База класстын мүчөсүнө кайрылуу  
object.korsot\_j();// Туунду класстын мүчөсүнө кайрылуу  
return 0;  
}

*basa* классы ачык класс катары мурасталгандан кийин, ал класстын ачык мүчөлөрү болгон *tanda\_i()* жана *korsot\_i()* функциялары туунду *tuundu* класс үчүн да ачык болуп, аларга програмдын каалаган бөлүгүнөн кайрылсак болот экен.

Бирок туунду класс үчүн база класстын жабык мүчөлөрү жабык болот. Жогорку програмды өзгөртүп төмөнкүчө жазсак ката жөнүндө билдирүү алабыз.

#include <iostream>  
using namespace std;  
  
//база классты аныктоо  
class basa{  
int i;  
public:  
void tanda\_i(int x){i=x;}  
void korsot\_i() {cout<<i<<”\n”;}  
};  
  
//Класс ачык класс катары мурасталат  
class tuundu:public basa{  
int j;  
public:  
void tanda\_j(int y){j=y;}  
void korsot\_j() {cout<<j<<”\n”;}  
void summa(){cout<<i+j<<”\n”;} //Ката  
};

Мында туунду класста база класстын жабык болгон *i* өзгөрмөсүнө кайрылуу аракети жасалып жатат. Бул туура эмес, себеби мурас алуу жолуна карабай, база класстын жабык мүчөлөрү ар дайым жабык болот.

Төмөнкү програмда база класс жабык класс катары мурасталат, б.а. *private* кызматчы сөз аркылуу. Мындай өзгөртүү компилөө учурунда ката жөнүндө маалымат берет. Андай каталар комментарийлерде жазылган.

#include <iostream>  
using namespace std;  
  
//база классты аныктоо  
class basa{  
int i;  
public:  
void tanda\_i(int x){i=x;}  
void korsot\_i() {cout<<i<<”\n”;}  
};  
  
//Класс жабык класс катары мурасталат  
class tuundu:private basa{  
int j;  
public:  
void tanda\_j(int y){j=y;}  
void korsot\_j() {cout<<j<<”\n”;}  
};  
  
int main()  
{  
tuundu object;  
object.tanda\_i(3); //Ката. База класстын мүчөсү жабык  
object.tanda\_j(5);// Туунду класстын мүчөсүнө кайрылуу  
object.korsot\_i();//Ката. База класстын мүчөсү жабык   
object.korsot\_j();// Туунду класстын мүчөсүнө кайрылуу  
return 0;  
}

Бирок база класстын *tanda\_i()* жана *korsot\_i()* функциялары туунду класс кандайча мурас алганына карабай, ачык болот. Демек *basa* тибиндеги каалаган объект програмдын каалаган жеринен ал функцияларга кайрыла алат. Мисалы төмөнкү кайрылуу туура болот:

basa object1;  
object1. tanda\_i(9);//туура, себеби object1 объектиси basa тибинде

База класс *private* кызматчы сөз аркылуу мурас алса, туунду класстын объектери үчүн база класстын мүчөлөрү жабык болот. Бирок туунду класстын ичинде ал мүчөлөр ачык болот. Мисалда жогорку програмдын өзгөртүлгөн варианты берилген.

#include <iostream>  
using namespace std;  
  
//база классты аныктоо  
class basa{  
int i;  
public:  
void tanda\_i(int x){i=x;}  
void korsot\_i() {cout<<i<<”\n”;}  
};  
  
//Класс жабык класс катары мурасталат  
class tuundu:private basa{  
int j;  
public:  
//tanda\_i() функциясына кайрылса болот  
void tanda\_i\_j(int y, int z){tanda\_i(y);j=z;}  
//korsot\_i() функциясына кайрылса болот  
void korsot\_i\_j() { korsot\_i(); cout<<j<<”\n”;}  
};  
  
int main()  
{  
tuundu object;  
object.tanda\_i\_j(3,5);  
object.korsot\_i\_j();  
return 0;  
}

Демек туунду класс ичинде *tanda\_i()*жана*korsot\_i()*функцияларына кайрылса болот экен.

**Класстын корголгон мүчөлөрү**

Туунду класс база класстын жабык мүчөлөрүнө кирүү мүмкүндүгү жок. Эгерде туунду класстан база класстын мүчөлөрүнө кайрылууга туура келсе, ал мүчөлөр ачык деп жарыяланыш керек. Бирок кээ бир учурларда база класстын жабык мүчөлөрүнө туунду класстан кайрылууга туура келет. Бул ойду ишке ашыруу үчүн Си++тилинде *рrotected* (корголгон) спецификатору колдонот.

Класс мындай спецификатор аркылуу байандалган учурда класстын корголгон мүчөлөрү ал класстын бардык туунду класстары үчүн кирүүгө мүмкүндүк берет. База жана туунду класстардын сыртында корголгон мүчөлөргө кайрылууга мүмкүн эмес.

Кирүүгө мүмкүндүк берген *рrotected* спецификатору жарыяланып жаткан класстын каалаган жеринде жайгашышы мүмкүн. Бирок, негизинен, аны класстын жабык мүчөлөрүн жарыялагандан кийин жайгаштырат. Төмөндө классты жарыялоонун негизги калыбынын толугу берилди:

class класстын\_ысымы{  
//жабык мүчөлөр  
protected:  //жок болушу мүмкүн  
//корголгон мүчөлөр  
public:  
//ачык мүчөлөр  
};

База класс туунду класска ачык *(publiс)* түрүндө мурасталса, база класстын корголгон мүчөсү туунду класста корголгон мүчө болот. База класс туунду класска жабык (*private***)** түрүндө мурасталса, база класстын корголгон мүчөсү туунду класс үчүн жабык мүчө болот. База класс туунду класска корголгон (*рrotected***)** түрүндө мурасталса, база класстын ачык жана корголгон мүчөлөрү туунду класс үчүн корголгон мүчөлөр болот.

Төмөнкү програмда класстын ачык, жабык жана корголгон мүчөлөрүнө кирүү мүмкүнчүлүктөрү каралган.

#include <iostream>  
using namespace std;  
  
class misal{  
//класстын жабык мүчөлөрү  
int i;  
protected: //булар дагы класстын жабык мүчөлөрү  
int j;  
public:  
int z;  
misal(int x, int y){i=x; j=y;}  
int aluu\_i(){return i;}  
int aluu\_j(){return j;}  
};  
  
int main()  
{  
misal object(3,5);  
/\*object.j=99; Ката.j өзгөрмөсү корголгон мүчө, ошондуктан жабык\*/  
object.z=101; //Туура. z өзгөрмөсү класстын ачык мүчөсү  
cout<<object.aluu\_i()<<”\n”;  
cout<<object.aluu\_j()<<”\n”;  
cout<<object.z;  
return 0;  
}

*main()* функциясынын ичинде *j* өзгөрмөсүнө кайрылууга мүмкүн эмес, себеби ал корголгон мүчө болгондуктан *misal* классы үчүн жабык.

Төмөнкү мисалда класстын корголгон мүчөлөрү ачык мүчө катары мурасталган учуру каралды.

#include <iostream>  
using namespace std;  
  
class basa{  
//класстын жабык мүчөлөрү  
protected: //булар класстын жабык мүчөлөрү  
int i, j;//бирок туунду класс үчүн кирүүгө мүмкүндүк бар  
public:  
void tandoo\_i\_j(int x, int y){i=x; j=y;}  
};  
  
class tuundu:public basa{  
int k;  
public:  
void tandoo\_k(int z){k=z;}  
/\*Төмөнкү функция үчүн basa классынын i жана j өзгөрмөлөрүнө   
кирүү мүмкүндүгү бар\*/  
void korsot\_i\_j\_k(){cout<<i<<’ ‘<<j<<’ ‘<<k;}  
};  
  
int main()  
{  
tuundu object;  
object.tandoo\_i\_j(3,5);  
object.tandoo\_k(10);  
object.korsot\_i\_j\_k();  
return 0;  
}

*i* жана *j* өзгөрмөлөрү *basa* классында корголгон мүчө болгондуктан, алар туунду класска ачык мүчө катары мурасталат жана туунду класстын мүчө-функцияларында колдонот. Бирок бул класстардын сыртында аталган өзгөрмөлөр жабык жана аларга кирүү мүмкүндүгү жок.

Эгерде база класс корголгон класс катары мурасталса, анда база класстын ачык жана корголгон мүчөлөрү туунду класс үчүн корголгон мүчө болуп саналат. Жогорку мисалды өзгөртүп, *basa* классын корголгон класс катары мурастайлы.

#include <iostream>  
using namespace std;  
  
class basa{  
//класстын жабык мүчөлөрү  
protected: //булар класстын жабык мүчөлөрү  
int i, j;//бирок туунду класс үчүн кирүүгө мүмкүндүк бар  
public:  
void tandoo\_i\_j(int x, int y){i=x; j=y;}  
};  
  
class tuundu:protected basa{  
int k;  
public:  
void tandoo\_k(int z){k=z;}  
/\*Төмөнкү функция үчүн basa классынын i жана j өзгөрмөлөрүнө   
кирүү мүмкүндүгү бар\*/  
void korsot\_i\_j\_k(){cout<<i<<’ ‘<<j<<’ ‘<<k;}  
};  
  
int main()  
{  
tuundu object;  
object.tandoo\_i\_j(3,5);/\*Ката. tandoo\_i\_j() функциясы basa классынын корголгон мүчөсү\*/  
object.tandoo\_k(10);  
object.korsot\_i\_j\_k();  
return 0;  
}

Комментарийде көрсөтүлгөндөй, бул програм компилөө учурунда ката жөнүндө билдирүү берет, себеби *basa* классы корголгон класс катары мурасталган үчүн, анын ачык жана корголгон мүчөлөрү туунду класстын ичинде корголгон мүчө болуп, аларга *main()* функциясынын ичинен кайрылууга мүмкүн эмес.

**Конструкторлор, деструкторлор жана мурастоо**

База класс, туунду класс же эки класс тең конструктор же/жана деструкторго ээ болушу мүмкүн. Эгерде база жана туунду класстарда конструкторлор жана деструкторлор бар болсо, анда конструкторлор мурас алуу тартибинде, ал эми деструкторлор тескер тартибинде аткарылат. Демек, база класстын конструктору биринчи аткарылат экен, себеби база класс туунду класс бар экендиги жөнүндө «билбейт», жана база класс туунду класста аткарылган алдын-ала аныктоонун негизи болушу мүмкүн. Туунду класстын деструктору база класстын деструкторунан биринчи аткарылат, себеби туунду класс база класстын негизинде жатат. Андан сырткары, эгерде база класстын деструктору туунду класстын деструкторунан биринчи аткарылса, анда ал туунду класстын кыйроосуна (апат болушуна) алып келмек.

Төмөнкү програмда база жана туунду класстар үчүн конструктор жана деструкторлордун аткаруу ирети келтирилген.

#include <iostream>  
using namespace std;  
  
class basa{  
public:  
basa(){cout<<”База класстын конструктору \n”;}  
~basa(){cout<<”База класстын деструктору \n”;}  
};  
  
class tuundu:public basa{  
public:  
tuundu(){cout<<”Туунду класстын конструктору \n”;}  
~tuundu(){cout<<”Туунду класстын деструктору \n”;}  
};  
  
int main()  
{  
tuundu object;  
return 0;  
}

Програм төмөнкү жыйынтыкты берет:

База класстын конструктору   
Туунду класстын конструктору   
Туунду класстын деструктору   
База класстын деструктору

Конструкторлор мурастоо иретинде, ал эми деструкторлор тескер иретинде аткарылды.

Буга чейинки мисалдарда биз база жана туунду класстардын конструкторлоруна аргументтерди берүү маселесин караган жокбуз. Бирок аны ишке ашырса болот. Алдын-ала аныктоо туунду класста жүргүзүлгөндө, аргументтер кадимкидей жол менен берилет. Бирок база класска аргументтерди берүү учурунда, маселе таталыраак болот, себеби база жана туунду класстардын керектүү аргументтери туунду класстын конструкторуна берилет. Андан кийин, туунду класстын конструкторун жарыялоосунун кеңейтилген калыбы колдонулуп, тиешелүү аргументтер база класска берилет. Аргументтерди туунду класстан база класска берүү синтаксиси төмөнкүдөй:

туунду\_класстын\_конструктору(арг\_тизмеси):база\_класс(арг\_тизмеси)  
{  
//туунду класстын конструкторунун денеси  
}

База жана туунду класстар үчүн ошол эле бир аргументти колдонсо болот. Андан сырткары, туунду класс үчүн бардык аргументтерге көңүл салбай, түздөн-түз база класска жиберсе болот.

Төмөнкү мисалда туунду класстын конструкторуна аргумент берүү маселеси каралган.

#include <iostream>  
using namespace std;  
  
class basa{  
public:  
basa(){cout<<”База класстын конструктору \n”;}  
~basa(){cout<<”База класстын деструктору \n”;}  
};  
  
class tuundu:public basa{  
int i;  
public:  
tuundu(int x){  
cout<<”Туунду класстын конструктору \n”;  
i=x;}  
~tuundu(){cout<<”Туунду класстын деструктору \n”;}  
void korsot\_i(){cout<<i<<”\n”;}  
};  
  
int main()  
{  
tuundu object(5);  
object.korsot\_i();  
return 0;  
}

Төмөнкү мисалда база жана туунду класстарда аргументтер бар. Эки учурда тең конструкторлор ошол эле бирдей аргументтерди колдонот жана туунду класс түздөн-түз ал аргументти база класска жиберет.

#include <iostream>  
using namespace std;  
  
class basa{  
int i;  
public:  
basa(int x){  
cout<<”База класстын конструктору \n”;  
i=x;}  
~basa(){cout<<”База класстын деструктору \n”;}  
void korsot\_i(){cout<<i<<”\n”;}  
};  
  
class tuundu:public basa{  
int j;  
public:  
tuundu(int x):basa(x){  
cout<<”Туунду класстын конструктору \n”;  
j=x;}  
~tuundu(){cout<<”Туунду класстын деструктору \n”;}  
void korsot\_j(){cout<<j<<”\n”;}  
};  
  
int main()  
{  
tuundu object(5);  
object.korsot\_i();  
object.korsot\_j();  
return 0;  
}

Негизинен база жана туунду класстардын конструкторлору көпчүлүк учурларда бир эле аргументти колдонбойт. Ар бир класстын конструкторуна бир же бир нече аргумент берүү талабы чыгат. Аны ишке ашыруу үчүн туунду класстын конструкторуна эки класстын конструкторлоруна тең керек болгон аргументтер берилет. Андан кийин туунду класстын конструктору база класстын конструкторуна керек болгон аргументтерди жиберет. Төмөнкү програмда туунду класстын конструкторуна бир аргумент, база класстын конструкторуна башка аргумент кандайча берилгени көрсөтүлдү.

#include <iostream>  
using namespace std;  
  
class basa{  
int i;  
public:  
basa(int x){  
cout<<”База класстын конструктору \n”;  
i=x;}  
~basa(){cout<<”База класстын деструктору \n”;}  
void korsot\_i(){cout<<i<<”\n”;}  
};  
  
class tuundu:public basa{  
int j;  
public:  
tuundu(int x, int y):basa(y){  
cout<<”Туунду класстын конструктору \n”;  
j=x;}  
~tuundu(){cout<<”Туунду класстын деструктору \n”;}  
void korsot\_j(){cout<<j<<”\n”;}  
};  
  
int main()  
{  
tuundu object(3,5);  
object.korsot\_i();  
object.korsot\_j();  
return 0;  
}

База класска берилүүчү аргументти туунду класстын конструктору аркылуу кандайдыр бир жол менен аргументти иштетүүнүн зарылдыгы жок. Эгерде туунду класска ал аргумент кереги жок болсо, анда ал аны менен эч кандай амал жүргүзбөй, жөн эле база класска жөнөтөт. Төмөнкү мисалда *х* параметри туунду класстын конструкторунда колдонулбайт, ошондуктан ал жөн эле база класстын конструкторуна жөнөтүлөт.

#include <iostream>  
using namespace std;  
  
class basa{  
int i;  
public:  
basa(int x){  
cout<<”База класстын конструктору \n”;  
i=x;}  
~basa(){cout<<”База класстын деструктору \n”;}  
void korsot\_i(){cout<<i<<”\n”;}  
};  
  
class tuundu:public basa{  
int j;  
public:  
tuundu(int x):basa(х){  
cout<<”Туунду класстын конструктору \n”;  
j=99;}  
~tuundu(){cout<<”Туунду класстын деструктору \n”;}  
void korsot\_j(){cout<<j<<”\n”;}  
};  
  
int main()  
{  
tuundu object(3);  
object.korsot\_i();  
object.korsot\_j();  
return 0;  
}

**Көпчүлүк мурастоо**

Туунду класс бир нече база класстардан мурас алышы мүмкүн. Андай ыкманын эки жолу бар. Биринчиси, туунду класс башка туунду класска база класс катары колдонулушу мүмкүн. Бул учурда баштапкы классты туунду класска карата ***кыйыр база классы***деп аталат. Экинчиси, туунду класс түздөн-түз бир нече класстан мурас алышы мүмкүн. Мындай учурларда туунду классты түзүүгө бир же бир нече база класстын аралашуусу жардам берет.

Эгерде туунду класс башка класска база класс катары колдонулса, анда үч класстын конструкторлору мурас берүү иретинде чакырылат. Деструкторлор тескер тартибинде чакырылат.

Туунду класс түздөн-түз бир нече база класстан мурас алса, анда төмөнкүдөй кеңейтилген жарыядоо колдонулат:

class туунду\_класстын\_ысымы:

 жетүү\_ыкмасы 1\_база\_класстын\_ысымы,

жетүү\_ыкмасы 2\_база\_класстын\_ысымы,  
жетүү\_ыкмасы … N\_база\_класстын\_ысымы  
{  
//… класстын денеси  
}

***1\_база\_класстын\_ысымы,2\_база\_класстын\_ысымы,…N\_база\_класстын\_ысымы*** база класстардын ысымдары, ***жетүү\_ыкмасы*** жетүү спецификатору. Жетүү спецификатору ар кандай база класстар үчүн ар кандай болушу мүмкүн. Бир нече база класс мурасталган учурда, конструкторлор солдон оңго карай жазылган тартибинде аткарылат. Деструкторлор тескер тартипте аткарылат.

Эгерде мурас берген класстардын конструкторлоруна аргументтер талап кылынса, туунду класс ал аргументтерди туунду класстын конструкторун жарыялоосунун кеңейтилген калыбын колдонот:

туунду\_класс\_конструкт(арг.тизм.):   1\_база\_класстын\_ысымы(арг.тизм.),  
                                    2\_база\_класстын\_ысымы(арг.тизм.),  
                             …      N\_база\_класстын\_ысымы(арг.тизм.)  
{  
//туунду класстын конструкторунун денеси  
}

Төмөнкү мисалда туунду класс башка класска карата туунду болгон класстан мурас алат.

#include <iostream>  
using namespace std;  
  
class klass1{  
int i;  
public:  
klass1(int x){i=x;}  
int aluu\_i(){return i;}  
};  
  
//База класстан түздөн-түз мурас алуу  
class klass2:public klass1{  
int j;  
public:  
klass2(int x, int y):klass1(y) //y өзгөрмөсүн klass1

                              //классына жиберүү  
{  
j=x;  
}  
int aluu\_j(){return j;}  
};  
  
//Туунду классты түздөн-түз мурастоо  
//жана база классты кыйыр мурастоо  
class klass3:public klass2{  
int k;  
public:  
klass3(int x, int y, int z):klass2(y,z) /\*аргументтерди klass2 классына жиберүү\*/  
{  
k=z;  
}  
void korsot()  
{  
cout<< aluu\_i()<<’ ‘<<aluu\_j()<<’ ‘<<k<<”\n”;  
}  
};  
  
int main()  
{  
klass3 object(3,4,5);  
object.korsot();  
// aluu\_i() жана aluu\_j() функциялары бул жерде да ачык  
cout<< object.aluu\_i()<<’ ‘<<object.aluu\_j();  
return 0;  
}

Програмда класстардын иерархынын ар бир классы мурдагы база класска бардык аргументтерди берип турат. Аталган класстардын иерархы төмөнкү сүрөттө көрсөтүлгөн:

Төмөнкү програмда туунду класс түздөн-түз эки класстан мурас алат:

#include <iostream>  
using namespace std;  
  
//Биринчи база классты түзүү  
class klass1{  
int i;  
public:  
klass1(int x){i=x;}  
int aluu\_i(){return i;}  
};  
  
//Экинчи база классты түзүү  
class klass2{  
int j;  
public:  
klass2(int x){j=x;}  
int aluu\_j(){return j;}  
};  
  
//Эки класстан түздөн-түз мурас алуу  
class klass3:public klass1, public klass2{  
int k;  
public:  
/\*аргументтерди түздөн-түз klass1 жана klass2 класстарына жиберүү\*/  
klass3(int x, int y, int z):klass1(z), klass2(y)   
{k=х;}  
void korsot()  
{  
cout<< aluu\_i()<<’ ‘<<aluu\_j()<<’ ‘<<k<<”\n”;  
}  
};  
  
int main()  
{  
klass3 object(3,4,5);  
object.korsot();  
return 0;  
}

Програмдын бул вариантында *klass3* классы *klass1* жана *klass2* класстарына аргументтерди өз-өзүнчө бөлүп берет. Бул учурда класстардын иерархы төмөнкүдөй болот:

Төмөнкү програмда туунду класс бир нече база класстан мурас алган учурда конструкторлор жана деструкторлордун чакыруу ирети көрсөтүлгөн.

#include <iostream>  
using namespace std;  
  
class klass1{  
public:  
klass1(){cout<<”klass1 классынын конструктору \n”;}  
~klass1(){cout<<”klass1 классынын деструктору \n”;}  
};  
  
class klass2{  
public:  
klass2(){cout<<”klass2 классынын конструктору \n”;}  
~klass2(){cout<<”klass2 классынын деструктору \n”;}  
};  
  
class klass3: public klass1, public klass2{  
public:  
klass3(){cout<<”klass3 классынын конструктору \n”;}  
~klass3(){cout<<”klass3 классынын деструктору \n”;}

};  
  
int main()  
{  
klass3 object;  
return 0;  
}

Бул програм төмөнкү жыйынтыкты берет:

klass1 классынын конструктору   
klass2 классынын конструктору   
klass3 классынын конструктору   
klass3 классынын деструктору   
klass2 классынын деструктору   
klass1 классынын деструктору

**Виртуал база класстар**

Эгерде база класс туунду класс тарабынан бир нече жолу мурасталса, анда проблем пайда болушу мүмкүн. Ал проблемди түшүнүү үчүн, класстардын төмөнкү иерархына назар салалы:

Мында *База класс* база классы *Туунду класс\_1* жана *Туунду класс\_2* туунду класстар менен мурасталат. *Туунду класс\_3* туунду классы түздөн-түз *Туунду класс\_1* жана *Туунду класс\_2* туунду класстардан мурас алат. Демек *База класс* база классы*Туунду класс\_3* туунду классы тарабынан *Туунду класс\_1* жана *Туунду класс\_2* туунду класстар аркылуу эки жолу мурасталат экен. Эгерде *База класс*база классы *Туунду класс\_3* туунду классында колдонсо, анда база класстын эки көчүрмөсү түзүлөт. Мындай көп маанилүүлүктү жойу үчүн Си++ тилинде*Туунду класс\_3* туунду классына *База класс* база класстын бир гана көчүрмөсүн кошуу механизми иштелип чыккан. Андай механизми *виртуал база класс* деп аталат. Туунду класс база класстан кыйыр мурас алса, мындай механизм каалаган туунду класстарда эки же андан көп көчүрмөлөрдүн пайда болушуна алып келбейт. Бул механизмди колдонуу үчүн база класска жетүү спецификатордун алдына *virtual* кызматчы сөзүн койуш керек.

Төмөнкү мисалда *tuundu\_3* классында *basa* классынын эки көчүрмөсү пайда болбош үчүн, виртуалдык база класс колдонот.

#include <iostream>  
using namespace std;  
  
class basa{  
public:  
int i;  
};  
  
class tuundu\_1:virtual public basa{  
public:  
int j;  
};  
  
class tuundu\_2:virtual public basa{  
public:  
int k;  
};  
  
class tuundu\_3:public tuundu\_1, public tuundu\_2{  
public:  
int summa(){return i+j+k;}  
};  
  
int main()  
{  
tuundu\_3 object;  
object.i=1;  
object.j=2;  
object.k=3;  
cout<<"Сумма="<< object.summa();  
return 0;  
}

Эгерде*tuundu\_1* жана *tuundu\_2* класстары *basa* классын виртуал классы катары мурастабаса, анда

object.i=1;

көрсөтмөсү көп маанилүүлүккө алып келмек.

Эгерде база класс туунду класс тарабынан виртуал класс катары мурасталса, анда туунду класстын ичинде база класстын көчүрмөсү түзүлөт. Жогорку мисалга карата төмөнкү көрсөтмө туура болот:

tuundu\_1.object;  
object.i=10;

Жөнөкөй жана виртуал база класстардын айырмачылыгы база класс бирден көп жолу мурасталганда сезилет.