ជំពូកទី ១១

អំពី **Polymorphism**

១. អំពី Polymorphism

លំនាំ OOP រួមមានធាតុសំខាន់ៗបី គឺ ៖ data abstraction និង encapsulation, inheritance និង polymorphism។ data abstraction និង encapsulation គឺជាដំណើរការនៃការបង្កើតប្រភេទទិន្នន័យថ្មី។ បញ្ហានេះទាក់ទងទៅនឹងការតាងផ្នែកខាងក្នុងនៃ data members និង ការប្រកាស methods, ការកំណត់លក្ខណៈនិងការប្រើនៅក្នុងកម្មវិធី។ Inheritance មានទំនាក់ទំនងទៅនឹងការបង្កើតប្រភេទទិន្នន័យថ្មីពីប្រភេទទិន្នន័យដែលមានរួចហើយក្នុងទម្រង់ជាលំដាប់ថ្នាក់។ យើងបានសិក្សារួចហើយអំពី data hiding, encapsulation និង inheritance នៅជំពូកមុនៗនេះ។

Poly មកពីពាក្យ Greek មានន័យថា many (ច្រើន) និង morphism មកពី Greek មានន័យថា form (ទម្រង់) ដូចនេះ polymorphism មានន័យថា many forms (ច្រើនទម្រង់)។ នៅក្នុង OOP, polymorphism សំដៅទៅលើ methods (ឬ member function) មានឈ្មោះដូចគ្នាដែលមានចរិតលក្ខណៈខុសគ្នា អាស្រ័យទៅលើប្រភេទនៃ object ដែលវាបញ្ជាក់។

Polymorphism គឺជាដំណើរការនៃការបង្កើត object មួយចំនួនរបស់ class ផ្សេងគ្នាទៅជាសំណុំមួយ ហើយការហៅ methods ដើម្បីអនុវត្តនូវការប្រតិបត្តិនៃ objects ដោយប្រើ function ផ្សេងគ្នា។ ម៉្យាងវិញទៀត polymorphism មានន័យថា "អនុវត្តនូវជំហានដំណើរការផ្សេងគ្នា តាមរយៈ functions ដែលមានឈ្មោះដូចគ្នា"។ វាចាត់ទុក objects នៃ class ដែលមានទំនាក់ទំនងគ្នាតាមលក្ខណៈទូទៅមួយ។ ពាក្យ virtual ត្រូវបានប្រើសម្រាប់ធ្វើសកម្មភាព polymorphism នៅក្នុង C++។ Polymorphism សំដៅទៅលើការភ្ជាប់ pointer មួយទៅនឹង method មួយនៅពេលគេ run។

២. អំពី Early binding // Complie Time;

ការជ្រើសយក function មួយតាមលក្ខណៈធម្មតានៅក្នុងខណៈពេលធ្វើការ compile នោះត្រូវបានហៅថា early binding ឬ static binding ឬ static linkage។ នៅក្នុងរយៈពេលធ្វើការ compile នោះ compiler របស់ C++ កំណត់នូវ function ដែលត្រូវប្រើ ដោយផ្អែកលើប៉ារ៉ាម៉ែត្រដែលត្រូវបញ្ជូនទៅឲ្យ function ឬ return type របស់ function។ រួចហើយ compiler ជំនួសត្រង់ function ដែលត្រូវនឹងសំណើនោះ។ compiler បែបនេះមានមូលដ្ឋានលើការជំនួសហៅថា static linkage។

តាមធម្មតា C++ អនុវត្តតាម early binding។ ជាមួយនឹង early binding នេះគេអាចសម្រេចបានដោយមានប្រសិទ្ធភាពច្រើន។ នៅក្នុងករណីនេះ ការហៅ function មានលក្ខណៈ លឿន ព្រោះគ្រប់ព័ត៌មានដែលត្រូវហៅ function មកប្រើនោះគឺមានតម្លៃស្ថិតក្នុងកម្មវិធីរួចជាស្រេចប្រសើរជាងការឲ្យតម្លៃបញ្ចូលតាម keyboard។

ភាសាដែលមានលក្ខណៈ OOP ពិតប្រាកដ ដូចជា Small talk ដែលអនុញ្ញាតឲ្យភ្ជាប់ methods នៅពេល run time តែប៉ុណ្ណោះ រីឯ C++ អាចឲ្យភ្ជាប់ methods ទាំង compile time និង run time។ តាមអត្ថន័យនេះ C++ គឺជាភាសាបង្កាត់ (hybrid) មួយដោយសារវាបង្កើតនូវ code ដែលមានលំនាំជា OOP ផង និងលំនាំជា procedural ផង។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមបង្ហាញការប្រតិបត្តិនៃ static binding។ កម្មវិធីនេះ គេ សរសេរឃ្លាសម្រាប់ចូលប្រើ methods របស់ derived class តាមរយៈ members នៃ base class ដោយប្រើវិធី pointer។

**// demonstration of static binding**

**#include <iostream.h>**

**class square {**

**protected :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ protected

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata()

**void display(); //** ការប្រកាសអនុគមន៍ display()

**int area(); //** ការប្រកាសអនុគមន៍ area()

**};**

**class rectangle:public square {**

**protected :**

**int y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ int មានលក្ខណៈ protected

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata()

**void display(); //** ការប្រកាសអនុគមន៍ display()

**int area(); //** ការប្រកាសអនុគមន៍ area()

**};**

**void square::getdata(){**

**cout << "Enter the value of side x ? \n";**

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**}**

**void square::display(){**

**cout << "Value of x = y = " << x << endl;**

**cout << "Area of the square = " << area() << endl;**

**}**

**int square::area(){**

**int temp = x \* x; //** ការធ្វើប្រមាណវិធីគុណ x និង x រួចកំណត់តម្លៃឲ្យអញ្ញាត temp

**return temp; //** ការឲ្យតម្លៃរបស់ temp ពេលហៅអនុគមន៍ area() មកប្រើ

**}**

**void rectangle::getdata(){**

**cout << "Enter the value of sides x and y ?\n";**

**cin >> x >> y; //** បញ្ចូលតម្លៃ x, y តាមរយៈ keyboard

**}**

**void rectangle::display(){**

**cout <<"Value of x = "<< x << " and y = "<<y <<endl;**

**cout << "Area of the rectangle = " << area() << endl;**

**}**

**int rectangle::area(){**

**int temp = x \* y; //** ការធ្វើប្រមាណវិធីគុណ x និង x រួចកំណត់តម្លៃឲ្យអញ្ញាត temp

**return temp; //** ការឲ្យតម្លៃរបស់ temp ពេលហៅអនុគមន៍ area() មកប្រើ

**}**

**void main(){**

**square sob; //** ការបង្កើត object នៃ square class

**rectangle rob; //** ការបង្កើត object នៃ rectangle class

**square \*ptr; //** ការបង្កើត object នៃ square class ជាប្រភេទ pointer

**ptr = &sob; //** ការកំណត់អស័យដ្ឋាននៃ object របស់ square class ទៅឲ្យ pointer ptr

**ptr = &rob; //** ការកំណត់អស័យដ្ឋាននៃ object របស់ rectangle class ទៅឲ្យ pointer ptr

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer នៃ square class

**ptr->area(); //** ការហៅអនុគមន៍ area() តាមរយៈ pointer នៃ square class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer នៃ square class

**}**

derived class ឈ្មោះ rectangle បានទទួលលក្ខណៈពី base class ឈ្មោះ square តាមរយៈ public derivation។ វាត្រូវបានគេចាត់ទុកថាជា object មួយរបស់ derived class មិនគ្រាន់តែទទួលលក្ខណៈពី base class ប៉ុណ្ណោះទេ ថែមទាំងមានលក្ខណៈលម្អិតបន្ថែមទៀតនៅក្នុង derived class។

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញនូវការប្រើឃ្លាដែលហៅពី derived class ទៅ base class objects។

**void main(){**

**square sob; //** ការបង្កើត object នៃ square class

**rectangle rob; //** ការបង្កើត object នៃ rectangle class

**square \*ptr; //** ការបង្កើត object នៃ square class ជាប្រភេទ pointer

**ptr = &sob; //** ការកំណត់អស័យដ្ឋាននៃ object របស់ square class ទៅឲ្យ pointer ptr

**ptr = &rob; //** ការកំណត់អស័យដ្ឋាននៃ object របស់ rectangle class ទៅឲ្យ pointer ptr

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer នៃ square class

**ptr->area(); //** ការហៅអនុគមន៍ area() តាមរយៈ pointer នៃ square class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer នៃ square class

**}**

object របស់ derived class អាចចូលប្រើបានតាមរយៈ pointer របស់ base class។ វិធី ប្រកាស និងការកំណត់តម្លៃរបស់ pointer ខាងក្រោមសម្រាប់ base class គឺអាចឲ្យគេសរសេរបានជាភាសា C++។

**square \*ptr; //** ការបង្កើត object នៃ square class ជាប្រភេទ pointer

**rectangle rob; //** ការបង្កើត object នៃ rectangle class

**ptr = &rob; //** ការកំណត់អស័យដ្ឋាននៃ object របស់ rectangle class ទៅឲ្យ pointer ptr

តាមរយៈ pointer នៃ base class នេះ data members និង member functions របស់ derived class អាចចូលទៅប្រើបាន។ ហេតុនេះឃ្លាសរសេរខាងក្រោមនេះមានលក្ខណៈត្រឹមត្រូវ។

**ptr->getdata();**

**ptr->area();**

**ptr->display();**

ការហៅអនុគមន៍មកប្រើត្រូវបានធ្វើឡើងតាមរយៈ pointer របស់ base class ដើម្បីចូលទៅប្រើ members របស់ derived class។ ទោះបីជាការហៅអនុគមន៍មកប្រើដោយត្រង់ៗដើម្បីចូលប្រើ members របស់ derived class ក៏ដោយ, ការហៅអនុគមន៍មកប្រើនេះមិនអាចចូលទៅដល់ members របស់ derived class បានឡើយ។ ហេតុនេះ members របស់ base class ត្រូវបានបង្ហាញឡើង តែវាមិនមែនជា members របស់ derived class ឡើយ។ នេះមកពី member functions របស់ base class និង derived class ប្រកាសឡើងដោយពុំមានលក្ខណៈជា virtual ហើយ compiler របស់ C++ ក៏ប្រើលក្ខណៈធម្មតាគឺ static binding។

C++ ផ្តល់នូវលក្ខណៈ polymorphism តាមរយៈ virtual methods និង pointers។ នៅក្នុងឧទាហរណ៍មុននេះ methods ត្រូវប្រកាសជា virtual នៅក្នុង base class ដូចខាងក្រោមនេះ។

**virtual void getdata();**

**virtual void display();**

**virtual int area();**

រីឯ derived class មាន methods ដូចគ្នាដដែលដោយវាមិនមែនជា virtual ឡើយ។

**void getdata();**

**void display();**

**int area();**

ចំពោះ functions ដែលមានឈ្មោះដូចគ្នាហើយមិនមែនជា virtual នោះប្រព័ន្ធកុំព្យូទ័រកំណត់យក នៅពេល compile សម្រាប់ functions ដែលត្រូវបានប្រើ។ ចំពោះ methods ដែលមានឈ្មោះដូចគ្នា ហើយមានលក្ខណៈជា virtual នោះ ប្រព័ន្ធកុំព្យូទ័រកំណត់យកនៅពេល run ចំពោះ methods ដែលត្រូវបាន ប្រើ។ methods ដែលប្រើត្រូវកំណត់តាមប្រភេទ object ដែល pointer បញ្ជាក់ប្រាប់។

ឧទាហរណ៍ បើ ptr កំពុងបញ្ជាក់ប្រាប់នូវ object នៃ square class នោះ members របស់ virtual methods នៃ square class ត្រូវបានយកមកប្រើដោយ ៖

**ptr->getdata(); // virtual methods of the base class**

**ptr->display(); // virtual methods of the base class**

**ptr->area(); // virtual methods of the base class**

បើ ptr កំពុងបញ្ជាក់ប្រាប់នូវ object នៃ rectangle class នោះ members របស់ rectangle class ត្រូវបានយកមកប្រើដោយឃ្លាដូចគ្នារបស់ function ដែលហៅប្រើ។

**ptr->getdata(); // methods of the derived class**

**ptr->display(); // methods of the derived class**

**ptr->area();​ // methods of the derived class**

ពេលនេះ គេអាចយល់បានថា polymorphism បង្កើតនូវ methods (member functions) ដែលមានឈ្មោះដូចគ្នាបេះបិទ ដោយមានចរិតលក្ខណៈខុសគ្នាតាមរយៈឃ្លាដូចគ្នា (ការហៅ អនុគមន៍មកប្រើ)។ ម៉្យាងវិញទៀត polymorphism សំដៅទៅលើការភ្ជាប់ pointer ទៅ method ក្នុងពេល run។ ពាក្យ virtual ត្រូវបានប្រើសម្រាប់អនុវត្តការភ្ជាប់ pointer មួយដែលបញ្ជាក់ទៅលើ member function នៃ object មួយក្នុងពេល run។

ឧទាហរណ៍ទី ២**៖** កម្មវិធីខាងក្រោមបង្ហាញនូវការភ្ជាប់ member functions នៃ class ពេល compile។ ឃ្លាដូចគ្នាដែលគេឲ្យសម្រាប់ចូលប្រើ member functions របស់ derived class តាមរយៈ array នៃ pointers។ ដោយប្រកាស functions មិនមានលក្ខណៈជា virtual នោះ compiler របស់ C++ ប្រើតាមវិធី static binding ប៉ុណ្ណោះ។

**// non-virtual function**

**// demonstration of compile time binding using**

**// array of pointers**

**#include <iostream.h>**

**class baseA {**

**public :**

**void display(){**

**cout << "One. \n"; //** បង្ហាញតម្លៃ One មកលើអេក្រង់

**}**

**};**

**class derivedB:public baseA {**

**public :**

**void display(){**

**cout << "Two. \n"; //** បង្ហាញតម្លៃ Two មកលើអេក្រង់

**}**

**};**

**class derivedC:public derivedB{**

**public :**

**void display(){**

**cout << "Three. \n"; //** បង្ហាញតម្លៃ Three មកលើអេក្រង់

**}**

**};**

**void main(){**

**// define three objects**

**baseA ob1; //** ការបង្កើត object នៃ baseA class

**derivedB ob2; //** ការបង្កើត object នៃ derivedB class

**derivedC ob3; //** ការបង្កើត object នៃ derivedC class

**baseA \*ptr[3]; //** ការបង្កើត pointer array នៃ baseA class ចំនួន 3

**ptr[0] = &ob1; //**ការកំណត់អាសយដ្ឋាននៃ object របស់ baseA class ទៅឲ្យ pointer ptr[0]

**ptr[1] = &ob2;//**ការកំណត់អាសយដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr[1]

**ptr[2] = &ob3;//**ការកំណត់អាសយដ្ឋាននៃ object របស់ derivedC class ទៅឲ្យ pointer ptr[2]

**for (int i = 0; i <= 2; i++)**

**ptr[i] -> display();// same message for all objects**

**}**

៣. អំពី Polymorphism ជាមួយនឹង pointers

pointers គឺជាស្នូលសម្រាប់ប្រើក្នុងលក្ខណៈ polymorphism របស់ភាសា C++។ ដើម្បីឲ្យមានលក្ខណៈ polymorphism បាននោះ C++ អនុញ្ញាត pointer នៅក្នុង base class មួយបញ្ជាក់ប្រាប់នូវ object របស់ base class ឬ derived class ណាមួយ។ code ខាងក្រោមនេះបង្ហាញនូវរបៀប pointer មួយដែលត្រូវកំណត់តម្លៃទៅឲ្យ object នៃ derived class។

**class baseA {**

**....**

**....**

**};**

**class derivedB:public baseA {**

**....**

**};**

**void main(){**

**baseA \*ptr; //** ការបង្កើត pointer នៃ baseA class

**derivedB obj; //** ការបង្កើត object នៃ derivedB class

**ptr=&obj; //**ការកំណត់អស័យដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr

**......**

**......**

**}**

pointer ptr បញ្ជាក់ប្រាប់នូវ object នៃ derived class ឈ្មោះ obj។

ផ្ទុយទៅវិញ pointer បញ្ជាក់ប្រាប់នូវ object នៃ derived class នោះអាចមិនបញ្ជាក់នូវ object នៃ base class បើសិនជាគ្មានការប្រើលក្ខណៈ casting កំណត់ឲ្យឃើញច្បាស់ទេនោះ។

ឧទាហរណ៍ code ខាងក្រោមនេះមិនត្រឹមត្រូវឡើយ ៖

**void main(){**

**baseA obj; //** ការបង្កើត object នៃ baseA class

**derivedB \*ptr; //** ការបង្កើត pointer នៃ derivedB class

**ptr = &obj; //** មានលក្ខណៈមិនត្រឹមត្រូវ

**......**

**......**

**}**

ចូរកត់សម្គាល់ថា pointer នៃ derived class មិនអាចបញ្ជាក់ទៅឲ្យ objects នៃ base class ឡើយ។ ក៏ប៉ុន្តែ code ខាងលើនេះអាចកែតម្រូវដោយប្រើលក្ខណៈ casting។

**void main(){**

**​ square sob; //** ការបង្កើត object នៃ square class

**rectangle \*ptr; //** ការបង្កើត pointer នៃ rectangle class

**ptr = (rectangle\*) &sob; // explicit casting**

**ptr->display();**

**......**

**......**

**}**

code ខាងក្រោមនេះបង្ហាញនូវការកំណត់តម្លៃផ្សេងគ្នានៃ pointer មួយចំពោះ object របស់ base class ដែលគេឲ្យទៅលើ object នៃ derived class។

ករណីទី ១ ៖ pointer របស់ base class អាចបញ្ជាក់ទៅឲ្យ object នៃ class ដូចគ្នាឬ derived class។

**class baseA {**

**....**

**....**

**};**

**class derivedB : public baseA {**

**....**

**....**

**};**

**void main(){**

**baseA obja; //** ការបង្កើត object នៃ baseA class

**derivedB objb; //** ការបង្កើត object នៃ derivedB class

**baseA \*ptr; //** ការបង្កើត pointer នៃ baseA class

**ptr = &obja; //** មានលក្ខណៈត្រឹមត្រូវ

**ptr = &objb; //** មានលក្ខណៈត្រឹមត្រូវ

**......**

**......**

**}**

ករណីទី ២ ៖ pointer របស់ derived class មិនអាចបញ្ជាក់ទៅឲ្យ object នៃ base class ឡើយ ក៏ប៉ុន្តែវាអាចបញ្ជាក់ទៅឲ្យ object នៃ class ដូចគ្នា។

**class baseA {**

**....**

**....**

**};**

**class derivedB : public baseA {**

**....**

**....**

**};**

**void main(){**

**baseA obja; //** ការបង្កើត object នៃ baseA class

**derivedB objb; //** ការបង្កើត object នៃ derivedB class

**​​ derivedB \*ptr; //** ការបង្កើត pointer នៃ derivedB class

**ptr=&obja; // error**

**ptr=&objb; // valid**

**......**

**......**

**}**

គួរកត់សម្គាល់ថា pointer របស់ derived class មិនអាចបញ្ជាក់ទៅឲ្យ object នៃ base class ឡើយ។

ឧទាហរណ៍ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវការកំណត់តម្លៃឲ្យ pointer របស់ derived class ទៅលើ object នៃ base class ដោយប្រើលក្ខណៈ casting។

**// demonstration of run time binding using explicit casting**

**#include <iostream.h>**

**class square {**

**protected :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ protected

**public :**

**virtual void getdata(); //** ប្រកាសអនុគមន៍ getdata() ជាលក្ខណៈ virtual

**virtual void display(); //** ប្រកាសអនុគមន៍ display() ជាលក្ខណៈ virtual

**virtual int area(); //** ប្រកាសអនុគមន៍ area() ជាលក្ខណៈ virtual

**};**

**class rectangle:public square {**

**protected :**

**int y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ int មានលក្ខណៈ protected

**public :**

**void getdata(); //** ប្រកាសអនុគមន៍ getdata()

**void display(); //** ប្រកាសអនុគមន៍ display()

**int area(); //** ប្រកាសអនុគមន៍ area()

**};**

**void square::getdata(){**

**cout << "Enter the value of side x :\n" ;**

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**}**

**void square::display(){**

**cout << "Value of x = y = " << x << endl;**

**cout << "Area of the square = " << area() << endl;**

**}**

**int square::area(){**

**int temp = x \* x; //** ការធ្វើប្រមាណវិធីគុណ x និង x រួចកំណត់តម្លៃឲ្យអញ្ញាត temp

**return temp; //** ការឲ្យតម្លៃរបស់ temp ពេលហៅអនុគមន៍ area() មកប្រើ

**}**

**void rectangle::getdata(){**

**cout << "Enter the value of sides x and y :\n";**

**cin >> x >> y; //** បញ្ចូលតម្លៃ x, y តាមរយៈ keyboard

**}**

**void rectangle::display(){**

**cout << "Value of x = " <<x <<" and y = " << y << endl;**

**cout << "Area of the rectangle = " << area() << endl;**

**}**

**int rectangle::area(){**

**int temp = x \* y; //** ការធ្វើប្រមាណវិធីគុណ x និង x រួចកំណត់តម្លៃឲ្យអញ្ញាត temp

**return temp; //** ការឲ្យតម្លៃរបស់ temp ពេលហៅអនុគមន៍ area() មកប្រើ

**}**

**void main(){**

**square sob; //** ការបង្កើត object នៃ square class

**rectangle \*ptr; //** ការបង្កើត pointer នៃ rectangle class

**ptr = (rectangle\*) &sob; //** explicit casting

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer នៃ rectangle class

**ptr->area(); //** ការហៅអនុគមន៍ area() តាមរយៈ pointer នៃ rectangle class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer នៃ rectangle class

**}**

៤. អំពីអនុគមន៍ដែលមានលក្ខណៈ Virtual (Virtual Functions)

Virtual function គឺជាអនុគមន៍ដែលមិនមានលក្ខណៈថិតថេរ ក៏ប៉ុន្តែវាបានលេចឡើងយ៉ាងពិតប្រាកដនៅក្នុងផ្នែកណាមួយនៃកម្មវិធី។ Virtual function គឺជាលក្ខណៈពិសេសបំផុតរបស់ OOP តែវាមិនមែនជាតម្រូវការសម្រាប់កម្មវិធី C++ ទាំងអស់ឡើយ។ នៅក្នុងផ្នែកនេះបង្ហាញនូវលក្ខណៈពិសេសនៃ polymorphism ដែលត្រូវធ្វើការដោយប្រើ virtual functions។

ទម្រង់ទូទៅនៃការប្រកាស virtual function គឺ ៖

*class user\_defined\_name {*

*private :*

*. . . .*

*. . . .*

*public :*

*virtual return\_type function\_name1(arguments);*

*virtual return\_type function\_name2(arguments);*

*virtual return\_type function\_name3(arguments);*

*. . . .*

*. . . .*

*}*

ដើម្បីបង្កើត member function មួយជា virtual នោះពាក្យ virtual ត្រូវបានប្រើជាមួយ methods ពេលដែលវាប្រកាសនៅក្នុងការបង្កើត class ក៏ប៉ុន្តែមិនត្រូវប្រើនៅពេលបង្កើត member function ឡើយ។ ពាក្យ virtual ត្រូវតែដាក់ពីមុខ return type នៃឈ្មោះ function។ compiler ទទួលព័ត៌មានពីពាក្យ virtual ថាវាជា virtual function ដែលមិនមែនជាការប្រកាស function ធម្មតាឡើយ។

ឧទាហរណ៍ ការប្រកាស virtual function ខាងក្រោមនេះមានលក្ខណៈត្រឹមត្រូវ។

**class sample {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void display(); //** ប្រកាសអនុគមន៍ display() ជាលក្ខណៈ virtual

**virtual int sum(); //** ប្រកាសអនុគមន៍ sum() ជាលក្ខណៈ virtual

**};**

ករណីខាងក្រោមនេះជាការប្រកាស virtual function ដែលមានលក្ខណៈមិនត្រឹមត្រូវនៅក្នុង ភាសា C++ ៖

ករណីទី ១ ៖ ពាក្យ virtual មិនត្រូវប្រើច្រំដែលនៅពេលប្រកាសក្នុង class និងនៅខាងក្រៅ class នៃការប្រកាសឡើយ។ ការប្រើ function មួយដែលមានបញ្ជាក់ពាក្យ virtual នៅក្នុងការបង្កើត function គឺ មិនត្រឹមត្រូវទេ។

**class sample {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void display(); //** ប្រកាសអនុគមន៍ display() ជាលក្ខណៈ virtual

**};**

**virtual void sample::display(){ // error**

**....**

**....**

**}**

ករណីទី ២ ៖ virtual function មិនអាចជា member ដែលមានលក្ខណៈ static បានទេ ព្រោះ virtual member ជានិច្ចជាកាលជា member របស់ object ពិសេសមួយនៅក្នុង class ដែលហាក់ដូចជា member នៃ class ទាំងមូល។

**class sample {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual static int sum(); //** error

**};**

**int sample::sum(){**

**....**

**....**

**}**

ករណីទី ៣ ៖ virtual function មិនអាចជា constructor member function បានឡើយ ក៏ប៉ុន្តែវាអាចកើតមាននៅ destructor member function។

**class sample {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual sample(int xx, float yy);//error constructor**

**void getdata(); //** ប្រកាសអនុគមន៍ getdata()

**void display(); //** ប្រកាសអនុគមន៍ display()

**};**

ចូរកត់សម្គាល់ថា វាមាន error កើតឡើងដោយ constructor មានលក្ខណៈជា virtual។

ករណីទី ៤ ៖ destructor member function មិនទទួលយកនូវ argument ណាមួយឡើយ ហើយពុំមាន return type ដែលត្រូវបញ្ជាក់ទៅអោយវា ទោះជា void ក៏គ្មានដែរ។

**class sample {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual ~sample(int xx, float yy); //** មិនត្រឹមត្រូវ

**void getdata(); //** ប្រកាសអនុគមន៍ getdata()

**void display(); //** ប្រកាសអនុគមន៍ display()

**};**

ចូរកត់សម្គាល់ថា destructor member function អាចជា virtual ទោះបីវាពុំមាន argument ណាមួយក៏ដោយ។

ករណីទី ៥**៖** វាមាន error កើតឡើង នៅពេលដែលការបង្កើត virtual method មានការប្តូរ return type នៅក្នុង derived class ជាមួយនឹងប្រភេទទិន្នន័យរបស់ប៉ារ៉ាម៉ែត្រដូចគ្នាទៅនឹងប្រភេទទិន្នន័យរបស់ប៉ារ៉ាម៉ែត្រនៃ virtual method នៅក្នុង base class ក៏ដោយ។

**class baseA {**

**private :**

**​​​ int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual int sum(int xx, float yy); //** មាន error កើតឡើង

**void getdata(); //** ប្រកាសអនុគមន៍ getdata()

**void display(); //** ប្រកាសអនុគមន៍ display()

**};**

**class derivedB:public baseA {**

**private :**

**int z; //** ប្រកាសអញ្ញាត z ជាប្រភេទ int មានលក្ខណៈ private

**public :**

**virtual float sum(int xx, float yy);**

**};**

ការប្រកាស virtual functions ទាំងពីរខាងលើមិនត្រឹមត្រូវឡើយ ទោះជា functions ទាំងនេះ មាន arguments ដូចគ្នាបេះបិទក៏ដោយ តែ return type របស់វាគឺខុសគ្នា។

**virtual int sum(int xx, float yy); //** ជារបស់ base class

**virtual float sum(int xx, float yy); // ​**ជារបស់derived class

functions ទាំងពីរខាងលើអាចសរសេរឲ្យមានប្រភេទទិន្នន័យជា int នៅក្នុង base class ក៏ដូចនៅក្នុង derived class។

**virtual int sum(int xx, float yy); //** ជារបស់ base class

**virtual int sum(int xx, float yy); // ​**ជារបស់derived class

ករណីទី ៦ ៖មានតែ member function នៃ class ប៉ុណ្ណោះដែលអាចប្រកាសជា virtual។ វានឹងមាន error កើតឡើងនៅពេលយើងប្រកាសវាជា non-member function របស់ class មួយ។

**virtual void display(){ // ​**មាន error **​**កើតឡើង ព្រោះវាមិនមែនជា member function

**....**

**}**

៥. អំពី Late binding // Run time;

ការជ្រើសយក functions ក្នុងពេលប្រតិបត្តិហៅថា late binding ឬ dynamic binding ឬ dynamic linkage។ Late binding ត្រូវបានប្រើតាមរយៈ virtual functions។ object នៃ class មួយត្រូវតែប្រកាស ទោះជា pointer បញ្ជាក់ឲ្យ class ឬ reference បញ្ជាក់ឲ្យ class ក៏ដោយ។

ឧទាហរណ៍ការប្រកាស virtual function ខាងក្រោមនេះបង្ហាញនូវរបៀប late binding ឬ run time binding ដែលអាចប្រើបាន ៖

**class baseA {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void display();//** ប្រកាសអនុគមន៍ display() មានលក្ខណៈជា virtual

**int sum(); //** ប្រកាសអនុគមន៍ sum()

**};**

**class derivedB:public baseA {**

**private :**

**int x; //** ប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**void display(); // virtual**

**int sum(); //** ប្រកាសអនុគមន៍ sum()

**};**

**void main(){**

**baseA \*ptr; //** ការបង្កើត pointer នៃ baseA class

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**ptr = &ob; //** ការកំណត់អាសយដ្ឋាន object នៃ baseA class ទៅឲ្យ pointer នៃ baseA class

**....**

**ptr->display(); // run time binding**

**ptr->sum(); // compile time binding**

**}**

ពាក្យ virtual ត្រូវតែដាក់ជាប់ return type នៃ member function បើសិន run time ត្រូវ បានភ្ជាប់។ ផ្ទុយទៅវិញ compile time binding នឹងមានឥទ្ធិពលតាមធម្មតា។ code នៅខាងលើនេះមានតែអនុគមន៍ display() ប៉ុណ្ណោះដែលត្រូវបានប្រកាសជា virtual នៅក្នុង base class រីឯ sum() មិនមែនជា virtual ឡើយ។ ទោះជាឃ្លាដែលគេឲ្យតាមរយៈ pointer នៃ base class បញ្ជាក់ទៅឲ្យ​ object នៃ derived class ក៏ដោយ វាមិនអាចចូលប្រើអនុគមន៍ sum() របស់ derived class ឡើយ ព្រោះវាបានប្រកាសជា non-virtual។ អនុគមន៍ sum() ធ្វើការ compile តែ static binding ប៉ុណ្ណោះ។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវ run time binding នៃ member functions របស់ class។ ឃ្លាដូចគ្នាដែលគេឲ្យនោះនឹងអាចចូលប្រើ member functions របស់ derived class តាមរយៈ array នៃ pointers។ ដោយសារ functions បានប្រកាសជា virtual នោះបង្កើតឡើងនូវ dynamic binding។

**// virtual function demonstration of run time binding using**

**// array of pointers**

**#include <iostream.h>**

**class baseA {**

**public :**

**virtual void display(){**

**cout << "One. \n"; //** បង្ហាញតម្លៃ One មកលើអេក្រង់

**}**

**};**

**class derivedB:public baseA {**

**public :**

**virtual void display(){**

**cout << "Two. \n"; //** បង្ហាញតម្លៃ Two មកលើអេក្រង់

**}**

**};**

**class derivedC:public derivedB{**

**public :**

**virtual void display(){**

**cout << "Three. \n"; //** បង្ហាញតម្លៃ Three មកលើអេក្រង់

**}**

**};**

**void main(){**

**// define three objects**

**baseA ob1; //** ការបង្កើត object នៃ baseA class

**derivedB ob2; //** ការបង្កើត object នៃ derivedB class

**derivedC ob3; //** ការបង្កើត object នៃ derivedC class

**baseA \*ptr[3]; //** ការបង្កើត pointer array នៃ baseA class ចំនួន 3

**ptr[0] = &ob1; //**ការកំណត់អាសយដ្ឋាននៃ object របស់ baseA class ទៅឲ្យ pointer ptr[0]

**ptr[1] = &ob2; //**ការកំណត់អាសយដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr[1]

**ptr[2] = &ob3; //**ការកំណត់អាសយដ្ឋាននៃ object របស់ derivedC class ទៅឲ្យ pointer ptr[2]

**for (int i = 0; i <= 2; i++)**

**ptr[i] -> display(); // same message for all objects**

**}**

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមបង្ហាញ static binding នៃ member functions របស់ class មួយ។

**// accessing member functions with pointers**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedB:public base {**

**private :**

**int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void base::getdata(){**

**cout << "Enter an integer :\n";**

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**cout << "Enter a real number :\n";**

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**void base::display(){**

**cout << "Entered numbers are x = " << x;**

**cout << " and y = " << y << endl;**

**}**

**void derivedB::getdata(){**

**cout << "ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**​​ cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID and student's name :\n";**

**cout << id << '\t' << name << endl;**

**}**

**void main(){**

**base \*ptr; //** ការបង្កើត pointer នៃ base class

**derivedB obj; //** ការបង្កើត object នៃ derivedB class

**ptr = &obj;//** ការកំណត់អាសយដ្ឋាន នៃ object នៃ derivedB class ទៅឲ្យ pointer នៃ base class

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer នៃ base class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer នៃ base class

**}**

ឧទាហរណ៍ទី ៣ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវ dynamic binding នៃ member functions របស់ class មួយ។

**// accessing member functions with pointers**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void base::getdata(){**

**cout << "Enter an integer:";//** បង្ហាញអក្សរ Enter an integer : មកលើអេក្រង់

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**cout << "Enter a real number:";**

**//** បង្ហាញអក្សរ Enter a real number : មកលើអេក្រង់

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**void base::display(){**

**cout << "Entered numbers are x = " << x ;**

**cout << " and y = " << y << endl;**

**}**

**void derivedB::getdata(){**

**cout << "Enter id number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name :\n";**

**cout << id << '\t' << name << endl;**

**}**

**void main(){**

**base \*ptr; //** ការបង្កើត pointer នៃ base class

**derivedB obj; //** ការបង្កើត object នៃ derivedB class

**ptr = &obj; //**ការកំណត់អាសយដ្ឋាន នៃ object នៃ derivedB class ទៅឲ្យ pointer នៃ base class

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer នៃ base class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer នៃ base class

**}**

ទោះបីជា virtual functions អាចប្រកាសជា inline code ក៏ដោយ ក៏វាត្រូវតែជា run time binding របស់ compiler ហើយ inline code ពុំមានឥទ្ធិពលច្រើនលើការសរសេរកម្មវិធីឡើយ។ ចំពោះ ការជំនួស inline code នោះ compiler ត្រូវតែទទួលព័ត៌មានពី function ដូចជាពីទីកន្លែងដែលវាបានយកមកប្រើ -ល- លក្ខណៈទាំងនេះត្រូវបានបង្កើតឡើងនៅពេលធ្វើការ compile។

ទម្រង់ទូទៅនៃ virtual function ជាមួយនឹងការជំនួស inline code គឺ ៖

**class base {**

**private :**

**....**

**....**

**public :**

**virtual inline void getdata();**

**virtual inline void display();**

**};**

**void main(){**

**base ob; //** ការបង្កើត object នៃ base class

**ob->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ object នៃ base class

**ob->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ object នៃ base class

**}**

ឧទាហរណ៍ទី ៤ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើត virtual functions ជាមួយនឹងការជំនួស inline code សម្រាប់ run time binding នៃ member functions របស់ class មួយ។

**// accessing member functions with pointers**

**// virtual functions with inline code**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual inline void getdata();**

**virtual inline void display();**

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**inline void base::getdata(){**

**cout << "Enter an integer:";//** បង្ហាញអក្សរ Enter an integer : មកលើអេក្រង់

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**cout << "Enter a real number:";**

**//** បង្ហាញអក្សរ Enter a real number : មកលើអេក្រង់

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**inline void base::display(){**

**cout << "Entered numbers are x = " << x ;**

**cout << " and y = " << y << endl;**

**}**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student ? \n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student ? \n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name \n";**

**cout << id << '\t' << name << endl;**

**}**

**void main(){**

**base \*ptr; //** ការបង្កើត pointer នៃ base class

**derivedB obj; //** ការបង្កើត object នៃ derivedB class

**ptr = &obj;//** កំណត់អាសយដ្ឋាន នៃ object នៃ derivedB class ទៅឲ្យ pointer នៃ base class

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer នៃ base class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer នៃ base class

**}**

ឧទាហរណ៍ទី ៥ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីការចូលប្រើ members នៃ derived class objects តាមរយៈ array នៃ pointers។ កម្មវិធីនេះមានតែ member functions របស់ base class ប៉ុណ្ណោះ ដែលត្រូវបានដាក់ពាក្យ virtual នៅពីមុខ។

**// accessing member functions with array of pointers**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedC:public base {**

**private :**

**float height; //** ការប្រកាសអញ្ញាត height ជាប្រភេទ float មានលក្ខណៈ private

**float weight; //** ការប្រកាសអញ្ញាត weight ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void base::getdata(){**

**cout << "Enter an integer:";//** បង្ហាញអក្សរ Enter an integer : មកលើអេក្រង់

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**cout << "Enter a real number:";**

**//** បង្ហាញអក្សរ Enter a real number : មកលើអេក្រង់

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**void base::display(){**

**cout << "Entered numbers are x = " << x ;**

**cout << " and y = " << y << endl;**

**}**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name : ";**

**cout << id << '\t' << name << endl;**

**}**

**void derivedC::getdata(){**

**cout << "Enter height of student :\n";**

**cin >> height; //** បញ្ចូលតម្លៃ height តាមរយៈ keyboard

**cout << "Enter weight of student :\n";**

**cin >> weight; //** បញ្ចូលតម្លៃ weight តាមរយៈ keyboard

**}**

**void derivedC::display(){**

**cout << "Height and weight of the student's :";**

**cout << height << '\t' << weight << endl;**

**}**

**void main(){**

**base \*ptr[3]; //** ការបង្កើត pointer នៃ base class ចំនួន 3

**derivedB ob1; //** ការបង្កើត object នៃ derivedB class

**derivedC ob2; //** ការបង្កើត object នៃ derivedC class

**ptr[0] = &ob1; //** កំណត់អាសយដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr[0]

**ptr[1] = &ob2; //** កំណត់អាសយដ្ឋាននៃ object របស់ derivedC class ទៅឲ្យ pointer ptr[1]

**ptr[0]->getdata();//** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr[0] នៃ base class

**ptr[1]->getdata();//** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr[1] នៃ base class

**ptr[0]->display();//** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr[0] នៃ base class

**ptr[1]->display();//** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr[1] នៃ base class

**}**

ឧទាហរណ៍ទី ៦ ៖ កម្មវិធីខាងក្រោមបង្ហាញពីការចូលប្រើ members នៃ derived class objects តាមរយៈ array នៃ pointers។ ក្នុងកម្មវិធីនេះទាំង member functions របស់ base class និង member functions របស់ derived class សុទ្ធតែបានដាក់ពាក្យ virtual នៅពីមុខ។

**// accessing member functions with array of pointers**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata();//**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display();//** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**virtual void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display();//** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedC:public base {**

**private :**

**float height; //** ការប្រកាសអញ្ញាត height ជាប្រភេទ float មានលក្ខណៈ private

**float weight; //** ការប្រកាសអញ្ញាត weight ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata();//**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display();//** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**void base::getdata(){**

**cout << "Enter an integer:";//** បង្ហាញអក្សរ Enter an integer : មកលើអេក្រង់

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**cout << "Enter a real number:";**

**//** បង្ហាញអក្សរ Enter a real number : មកលើអេក្រង់

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**void base::display(){**

**cout << "Entered numbers are x = " << x ;**

**cout << " and y = " << y << endl;**

**}**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name :";**

**cout << id << '\t' << name << endl;**

**}**

**void derivedC::getdata(){**

**cout << "Enter height of student :\n";**

**cin >> height; //** បញ្ចូលតម្លៃ height តាមរយៈ keyboard

**cout << "Enter weight of student :\n";**

**cin >> weight; //** បញ្ចូលតម្លៃ weight តាមរយៈ keyboard

**}**

**void derivedC::display(){**

**cout << "Height and weight of the student's :\n";**

**cout << height << '\t' << weight << endl;**

**}**

**void main(){**

**base \*ptr[3]; //** ការបង្កើត pointer នៃ base class ចំនួន 3

**derivedB ob1; //** ការបង្កើត object នៃ derivedB class

**derivedC ob2; //** ការបង្កើត object នៃ derivedC class

**ptr[0] = &ob1; //** កំណត់អាសយដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr[0]

**ptr[1] = &ob2; //** កំណត់អាសយដ្ឋាននៃ object របស់ derivedC class ទៅឲ្យ pointer ptr[1]

**ptr[0]->getdata();//** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr[0] នៃ base class

**ptr[1]->getdata();//** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr[1] នៃ base class

**ptr[0]->display();//** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr[0] នៃ base class

**ptr[1]->display();//** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr[1] នៃ base class

**}**

៦. អំពី Pure Virtual functions

Pure virtual function គឺជាប្រភេទ function ដែលមានតែការប្រកាស function ប៉ុណ្ណោះ។ វាមិនមាន function definition ឡើយ។ code ខាងក្រោមនេះបង្ហាញនូវរបៀបប្រកាស pure virtual function ៖

ករណីទី ១ ៖

**// pure virtual functions**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedB:public base {**

**.....**

**.....**

**};**

**void base::getdata(){ // pure virtual function definition**

**}**

**void base::display(){ // pure virtual function definition**

**}**

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមបង្ហាញ pure virtual function មួយបានបង្កើតឡើង, ការប្រកាស និងការហៅប្រើ object នៃ derived class តាមរយៈ pointer នៃ base class។

**// pure virtual functions**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata();//**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display();//**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void base::getdata(){ }**

**void base::display(){ }**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name :";**

**cout << id << '\t' << name << endl;**

**}**

**void main(){**

**base \*ptr; //** ការបង្កើត pointer នៃ base class

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**ptr = &ob; //** កំណត់អាសយដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr នៃ base class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr នៃ base class

**}**

ករណីទី ២ ៖ pure virtual function អាចមានទម្រង់ដូចខាងក្រោម នៅពេល virtual function ត្រូវបានប្រកាសឡើងក្នុង class ខ្លួនវា។ Virtual function អាចចាត់ទុកស្មើនឹងសូន្យ បើសិនវាមិនមាន function definition ទេនោះ។

**// pure virtual functions**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata()=0;**

**//** ការប្រកាសអនុគមន៍ getdata() ដោយកំណត់តម្លៃស្មើ 0

**virtual void display()=0;**

**//** ការប្រកាសអនុគមន៍ display() ដោយកំណត់តម្លៃស្មើ 0

**};**

**class derivedB:public base {**

**.....**

**.....**

**};**

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមបង្ហាញនូវរបៀបប្រកាស pure virtual function ហើយចាត់ទុកស្មើនឹងសូន្យ ព្រោះវាមិនមានផ្នែកណាមួយនៃ function ឡើយ។

**// pure virtual functions**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual inline void getdata() = 0;**

**virtual inline void display() = 0;**

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number student's name :";**

**cout << id << '\t' << name << endl;**

**}**

**void main(){**

**base \*ptr; //** ការបង្កើត pointer នៃ base class

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**ptr = &ob; //** កំណត់អាសយដ្ឋាននៃ object របស់ derivedB class ទៅឲ្យ pointer ptr

**ptr->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr នៃ base class

**ptr->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr នៃ base class

**}**

នៅពេល object នៃ derived class ព្យាយាមចូលទៅប្រើតាមរយៈ pointer នៃ base class members នោះ function ដែលហៅប្រើនឹងទៅដល់ members នៃ derived class មិនមែន base class members ទៀតឡើយ ព្រោះ member function របស់ base class ពុំមាន function definition។

៧. អំពី Abstract Base Class

Class មួយដែលមាន pure virtual functions ត្រូវបានហៅថា abstract base class។ នៅក្នុងផ្នែកមុនៗនេះ យើងបានពិភាក្សារួចហើយថា function មួយអាចបង្កើតឡើងដោយគ្មាន statement ណាមួយ ឬការប្រកាស functions អាចចាត់ទុកស្មើនឹងសូន្យ បើសិនវាពុំមាន function definition ទេ។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត abstract base class ដោយមាន pure virtual functions ដែលនៅក្នុងនោះ វាពុំមាន statement ណាមួយឡើយ។ members នៃ derived class objects ត្រូវបានចូលប្រើតាមរយៈ base class objects ដោយប្រើវិធី pointer។

**// Demonstration of abstract base class**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ virtual

**virtual void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedC: public base {**

**private :**

**float weight; //** ការប្រកាសអញ្ញាត weight ជាប្រភេទ float មានលក្ខណៈ private

**float height; //** ការប្រកាសអញ្ញាត height ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void base::getdata() {} // pure virtual function**

**void base::display() {} // pure virtual function**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name :";**

**cout << id << '\t' << name << endl;**

**}**

**void derivedC::getdata(){**

**cout << "Enter height of student :\n";**

**cin >> height; //** បញ្ចូលតម្លៃ height តាមរយៈ keyboard

**cout << "Enter weight of student :\n";**

**cin >> weight; //** បញ្ចូលតម្លៃ weight តាមរយៈ keyboard

**}**

**void derivedC::display(){**

**cout << "Height and weight of the student's :\n";**

**cout << height << '\t' << weight << endl;**

**}**

**void main(){**

**base \*ptr[3]; //** ការបង្កើត pointer array នៃ base class ចំនួន 3

**derivedB ob1; //** ការបង្កើត object នៃ derivedB class

**derivedC ob2; //** ការបង្កើត object នៃ derivedC class

**ptr[0] = &ob1;//** កំណត់អាសយដ្ឋាននៃ ob1 របស់ derivedB class ទៅឲ្យ pointer ptr[0]

**ptr[1] = &ob2;//** កំណត់អាសយដ្ឋាននៃ ob2 របស់ derivedC class ទៅឲ្យ pointer ptr[1]

**ptr[0]->getdata();//** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr[0] នៃ base class

**ptr[1]->getdata();//** ការហៅអនុគមន៍ getdata() តាមរយៈ pointer ptr[1] នៃ base class

**ptr[0]->display();//** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr[0] នៃ base class

**ptr[1]->display();//** ការហៅអនុគមន៍ display() តាមរយៈ pointer ptr[1] នៃ base class

**}**

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើត abstract base class ដោយមាន pure virtual functions ដែលក្នុងនោះការប្រកាស function ត្រូវបានចាត់ទុកស្មើនឹងសូន្យ ព្រោះថាវាមិនមាន function definition ឡើយ។

**// Demonstration of abstract base classes**

**// case 2**

**#include <iostream.h>**

**class base {**

**private :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ private

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**virtual inline void getdata()=0;**

**virtual inline void display()=0;**

**};**

**class derivedB:public base {**

**private :**

**long int id; //** ការប្រកាសអញ្ញាត id ជាប្រភេទ long int មានលក្ខណៈ private

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedC: public base {**

**private :**

**float weight; //** ការប្រកាសអញ្ញាត weight ជាប្រភេទ float មានលក្ខណៈ private

**float height; //** ការប្រកាសអញ្ញាត height ជាប្រភេទ float មានលក្ខណៈ private

**public :**

**void getdata(); //**ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void derivedB::getdata(){**

**cout << "Enter ID number of a student :\n";**

**cin >> id; //** បញ្ចូលតម្លៃ id តាមរយៈ keyboard

**cout << "Enter name of a student :\n";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**cout << "ID number and student's name :\n";**

**cout << id << '\t' << name << endl;**

**}**

**void derivedC::getdata(){**

**cout << "Enter height of student :\n";**

**cin >> height; //** បញ្ចូលតម្លៃ height តាមរយៈ keyboard

**cout << "Enter weight of student :\n";**

**cin >> weight; //** បញ្ចូលតម្លៃ weight តាមរយៈ keyboard

**}**

**void derivedC::display(){**

**cout << "Height and weight of the student's :\n";**

**cout << height << '\t' << weight << endl;**

**}**

**void main(){**

**base \*ptr[3]; //** ការបង្កើត pointer array នៃ base class ចំនួន 3

**derivedB ob1; //** ការបង្កើត object នៃ derivedB class

**derivedC ob2; //** ការបង្កើត object នៃ derivedC class

**ptr[0] = &ob1; //** កំណត់អាសយដ្ឋាននៃ ob1 របស់ derivedB class ទៅឲ្យ pointer ptr[0]

**ptr[1] = &ob2; //** កំណត់អាសយដ្ឋាននៃ ob2 របស់ derivedC class ទៅឲ្យ pointer ptr[1]

**ptr[0]->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ ptr[0] នៃ base class

**ptr[1]->getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ ptr[1] នៃ base class

**ptr[0]->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ ptr[0] នៃ base class

**ptr[1]->display(); //** ការហៅអនុគមន៍ display() តាមរយៈ ptr[1] នៃ base class

**}**

៨. Constructors នៅក្នុង Inheritance

នៅពេលដែល object នៃ class មួយបានបង្កើតឡើងនោះ constructor member function ត្រូវបានប្រើដោយស្វ័យប្រវត្ត ហើយនៅពេលដែល object នៃ derived class ត្រូវបានបង្កើតឡើង constructor នៃ object នោះនឹងត្រូវបានប្រើ។ នេះអាស្រ័យដោយ object នៃ derived class ដែល បានផ្ទុក members នៃ base class ផងដែរ។ ដោយសារតែ base class គឺជាផ្នែកមួយនៃ derived class នោះ តាមពិតទៅវាមិនសមហៅ constructors របស់ base class មកប្រើឡើយ។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើត​និងការប្រកាស constructor member function នៅក្នុង base class ក៏ដូចនៅក្នុង derived class ក្រោមឥទ្ធិពល inheritance។

**// Constructors under inheritance**

**#include <iostream.h>**

**class baseA {**

**public :**

**baseA(); //** ជា **constructor** របស់ baseA class

**};**

**class derivedB:public baseA {**

**public :**

**derivedB(); //** ជា **constructor** របស់ derivedB class

**};**

**baseA::baseA(){**

**cout << "Base class constructor. \n";**

**}**

**derivedB::derivedB(){**

**cout << "Derived class constructor. \n";**

**}**

**void main(){**

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**}**

កម្មវិធីខាងលើនេះមាន constructors ពីរ គឺ baseA() និង derivedB()។ នៅពេលដែល object ឈ្មោះ ob បានបង្កើតឡើងនោះ constructors របស់ base class បានហៅប្រើដោយស្វ័យប្រវត្ត។ ភាសា C++ ចាប់ផ្តើមប្រតិបត្តិពី class ទាបបំផុតទៅ class ខ្ពស់បំផុត។ មុននឹងបង្កើត derivedB() នោះ C++ ប្រតិបត្តិនូវ baseA() សិន។ ដូចនេះ constructor របស់ base class ប្រតិបត្តិមុន constructor របស់ derived class។

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើតនិងការប្រកាស constructor member function នៅក្នុង base class ក៏ដូចនៅក្នុង derived class ក្រោមឥទ្ធិពល inheritance។

**// Constructors under inheritance**

**#include <iostream.h>**

**class baseA {**

**public :**

**baseA(); //** ជា **constructor** របស់ baseA class

**};**

**class derivedB:public baseA {**

**public :**

**derivedB(); //** ជា **constructor** របស់ derivedB class

**};**

**class derivedC:public derivedB {**

**public :**

**derivedC(); //** ជា **constructor** របស់ derivedC class

**};**

**class derivedD:public derivedC {**

**public :**

**derivedD(); //** ជា **constructor** របស់ derivedD class

**};**

**baseA::baseA(){**

**cout << "baseA class.\n"; //** បង្ហាញអក្សរមួយឃ្លានេះមកលើអេក្រង់

**}**

**derivedB::derivedB(){**

**cout << "derivedB class.\n"; //** បង្ហាញអក្សរមួយឃ្លានេះមកលើអេក្រង់

**}**

**derivedC::derivedC(){**

**cout << "derivedC class.\n"; //** បង្ហាញអក្សរមួយឃ្លានេះមកលើអេក្រង់

**}**

**derivedD::derivedD(){**

**cout << "derivedD class.\n"; //** បង្ហាញអក្សរមួយឃ្លានេះមកលើអេក្រង់

**}**

**void main(){**

**derivedD ob; //** ការបង្កើត object នៃ derivedD class

**}**

៩. Destructors នៅក្នុង Inheritance

យើងបានដឹងហើយថា destructor គឺជា member function ពិសេសមួយ។ វាត្រូវបានប្រើដោយស្វ័យប្រវត្ត ដើម្បីធ្វើឲ្យទំនេរ memory ដែលបង្កើតឡើងដោយ constructor functions។ នៅពេល object នៃ class មួយត្រូវបានបំបាត់នោះ destructor ត្រូវបានប្រើ ដើម្បីធ្វើឲ្យទំនេរទីតាំង memory។ នៅផ្នែកមុនៗនេះ យើងបានដឹងថា constructors នៅក្នុងលំដាប់ថ្នាក់ inheritance ចាប់ផ្តើមប្រតិបត្តិការពី base class ទៅ derived class។ Destructors នៅក្នុង inheritance ចាប់ផ្តើមប្រតិបត្តិពី derived class ទៅ base class វិញ នេះបានន័យថាវា បញ្ច្រាសលំដាប់នៃ constructors។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពី destructor member function ចាប់ផ្តើមប្រតិបត្តិពី derived class objects ទៅក្នុង base class objects តាមរយៈ pointers។

**// Destructors under inheritance**

**#include <iostream.h>**

**class baseA {**

**public :**

**~baseA(); //** ជា **destructor** របស់ baseA class

**};**

**class derivedB:public baseA {**

**public :**

**~derivedB(); //** ជា **destructor** របស់ derivedB class

**};**

**baseA::~baseA(){**

**cout << "base class destructor\n";**

**}**

**derivedB::~derivedB(){**

**cout << "derivedB class destructor\n";**

**}**

**void main(){**

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**}**

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើត, ការប្រកាស និងហៅប្រើ destructor member function នៅក្នុងលក្ខណៈ multiple inheritance។

**// Destructors under inheritance**

**#include <iostream.h>**

**class baseA {**

**public :**

**~baseA(); //** ជា **destructor** របស់ baseA class

**};**

**class derivedB:public baseA {**

**public :**

**~derivedB(); //** ជា **destructor** របស់ derivedB class

**};**

**class derivedC:public derivedB {**

**public :**

**~derivedC(); //** ជា **destructor** របស់ derivedC class

**};**

**class derivedD:public derivedC {**

**public :**

**~derivedD(); //** ជា **destructor** របស់ derivedD class

**};**

**baseA::~baseA(){**

**cout << "baseA class.\n";**

**}**

**derivedB::~derivedB(){**

**cout << "derivedB class.\n";**

**}**

**derivedC::~derivedC(){**

**cout << "derivedC class.\n";**

**}**

**derivedD::~derivedD(){**

**cout << "derivedD class.\n";**

**}**

**void main(){**

**derivedD ob; //** ការបង្កើត object នៃ derivedD class

**}**

ឧទាហរណ៍ទី ៣ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញអក្សររបស់ constructors និង destructors នៃ base class ហើយនិង derived class។

**// Destructors under inheritance**

**#include <iostream.h>**

**class baseA {**

**public :**

**baseA(){ //** ជា **constructor** របស់ baseA class

**cout << "baseA class constructor\n";**

**}**

**~baseA(){ //** ជា **destructor** របស់ baseA class

**cout << "baseA class destructor\n";**

**}**

**};**

**class derivedB: public baseA {**

**public :**

**derivedB() { //** ជា **constructor** របស់ **derivedB**

**cout << "derivedB class constructor\n";**

**}**

**~derivedB(){ //** ជា **destructor** របស់ **derivedB**

**cout << "derivedB class destructor\n";**

**}**

**};**

**void main(){**

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**}**

ការងារចំបងរបស់ destructor គឺធ្វើឲ្យទំនេរទីតាំងផ្ទុករបស់កុំព្យូទ័រ។ ដោយការចាប់ ផ្តើមប្រតិបត្តិតាមលំដាប់បញ្ច្រាសនៃ constructor នោះ destructors ធានាបាននូវទីតាំងផ្ទុកថ្មីៗបំផុតឲ្យនៅទំនេរ។ កម្មវិធីខាងក្រោមនេះពន្យល់នូវរបៀប destructor member functions ធ្វើឲ្យទំនេរទីតាំង memory ដែលបានប្រើ new operator នោះ។

ឧទាហរណ៍ទី ៤ ៖ កម្មវិធីខាងក្រោមនេះបង្កើតទីតាំង memory ដោយប្រើ constructor member function និងធ្វើឲ្យទំនេរទីតាំងផ្ទុក memory ដោយ destructors នៅក្នុងលក្ខណៈ

multiple inheritance។

**// Destructors member function under inheritance**

**#include <iostream.h>**

**class baseA {**

**private :**

**char \*ptrbase; //** ការប្រកាសអញ្ញាត pointer ជាប្រភេទ char មានលក្ខណៈ private

**public :**

**baseA(); //** ជា **constructor** របស់ baseA class

**~baseA(); //** ជា **destructor** របស់ baseA class

**};**

**class derivedB: public baseA {**

**private :**

**char \*ptrderived;**

**public :**

**derivedB(); //** ជាconstructor របស់ derivedB class

**~derivedB(); //** ជា destructorរបស់ derivedB class

**};**

**baseA::baseA(){ //** ជា **constructor** របស់ baseA class

**ptrbase = new char[5];**

**cout << "Base class allocates 5 bytes \n";**

**}**

**baseA::~baseA(){ //** ជា **destructor** របស់ baseA class

**delete[] ptrbase;**

**cout << "Base class frees 5 bytes\n";**

**}**

**derivedB::derivedB(){ //** ជា **constructor** របស់ **derivedB**

**ptrderived = new char[100];**

**cout << "Derived class allocates 100 bytes \n";**

**}**

**derivedB::~derivedB(){ //** ជា **destructor** របស់ derivedB class

**delete[] ptrderived;**

**cout << "Derived class frees 100 bytes \n";**

**}**

**void main(){**

**derivedB ob; //** ការបង្កើត object នៃ derivedB class

**}**

១០. អំពី Virtual Destructors

យើងបានដឹងហើយថា destructor member function ត្រូវបានប្រើសម្រាប់ធ្វើឲ្យទំនេរ ទីតាំងផ្ទុកក្នុង memory ដោយស្វ័យប្រវត្ត។ ក៏ប៉ុន្តែ destructor member function របស់ derived class មិនបានប្រើសម្រាប់ធ្វើឲ្យទំនេរទីតាំងផ្ទុកដែលបង្កឡើងដោយ constructor member function របស់ derived class ឡើយ។ ព្រោះ destructor member functions មិនមែនជា virtual ហើយ message មិនអាចទៅដល់ destructor member function ក្រោមឥទ្ធិពល late binding បានឡើយ។ ហេតុនេះ គួរតែមាន destructor member function ជា virtual ហើយ virtual destructor គឺជាបញ្ហាសំខាន់នៅក្នុងកម្មវិធីសម្រាប់ធ្វើឲ្យទំនេរទីតាំងតាមវិធី late binding ដោយមានប្រសិទ្ធភាព។

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត virtual destructor ក្នុងកម្មវិធីមួយ។

**// virtual destructor**

**#include <iostream.h>**

**class baseA {**

**public :**

**baseA(); //** constructorមិនអាចដាក់ជាលក្ខណៈ virtual ទេ

**virtual ~baseA(); //** destructorមានលក្ខណៈជា virtual បាន

**};**

**class derivedB:public baseA {**

**....**

**....**

**};**

**baseA::~baseA(){ //** ជាdestructorរបស់ baseA class

**delete[] ptrbase;**

**cout << " base class frees 5 bytes \n";**

**}**

**void main(){**

**baseA \*ptr = new derivedB;//**ការបង្កើត object នៃ pointer របស់ derivedB class

**delete ptr; //** ការប្រើdelete operatorដើមី្បលុប ptr ពី memory

**}**

ចូរកត់សម្គាល់ថា នៅពេល object មួយត្រូវបានបង្កើតឡើងក្នុងពេល run time តាម រយៈការប្រើ new operator នោះ constructor member functions ត្រូវបានហៅប្រើដោយស្វ័យប្រវត្ត។ នៅពេល delete operator ត្រូវបានប្រើនោះ destructors ត្រូវបានប្រើដោយស្វ័យប្រវត្ត ដើម្បីធ្វើឲ្យទំនេរទីតាំងផ្ទុក object ខ្លួនវា។ ដោយសារ derived class instance ជានិច្ចជាកាលមានផ្ទុក base class instance នោះ វាចាំបាច់ត្រូវតែប្រើ destructors នៃ classes ទាំងពីរ ដើម្បីធានាដល់ការធ្វើឲ្យទំនេរទីតាំងដែលវាប្រើនោះ។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវទីតាំង memory ដែលបង្កើតឡើងដោយប្រើ new operator និងធ្វើឲ្យទំនេរ memory នៅពេល destructor member functions ត្រូវបានហៅប្រើ។

**// virtual destructor**

**#include <iostream.h>**

**class baseA {**

**private :**

**char \*ptrbase; //** ការប្រកាសអញ្ញាត pointer ជាប្រភេទ char មានលក្ខណៈ private

**public :**

**baseA(); //** constructorមិនអាចដាក់ជាលក្ខណៈ virtual ទេ

**virtual ~baseA(); //** destructorមានលក្ខណៈជា virtual បាន

**};**

**class derivedB:public baseA {**

**private :**

**char \*ptrderived; //**ការប្រកាសអញ្ញាត pointer ជាប្រភេទ char មានលក្ខណៈ private

**public :**

**derivedB(); //** constructorរបស់ derivedB class

**~derivedB(); //** destructorរបស់ derivedB class

**};**

**baseA::baseA(){ //** constructorរបស់ baseA class

**ptrbase = new char[5];**

**cout << "Base class allocates 5 bytes \n";**

**}**

**baseA::~baseA(){ //** destructorរបស់ baseA class

**delete[] ptrbase;**

**cout << "Base class frees 5 bytes\n";**

**}**

**derivedB::derivedB(){ //** constructorរបស់ derivedB class

**ptrderived = new char[100];**

**cout << "Derived class allocates 100 bytes \n";**

**}**

**derivedB::~derivedB(){ //** destructorរបស់ derivedB class

**delete[] ptrderived;**

**cout << "Derived class frees 100 bytes \n";**

**}**

**void main(){**

**baseA \*ptr = new derivedB;//**ការបង្កើត object នៃ pointer របស់ derivedB class

**delete ptr; //** ការប្រើdelete operatorដើមី្បលុប ptr ពី memory

**}**

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើត, ប្រកាស និងហៅប្រើ virtual destructor member function នៅក្នុងលក្ខណៈ multiple inheritance ដោយប្រើវិធី polymorphism។

**// virtual destructor function**

**#include <iostream.h>**

**class base {**

**public :**

**virtual void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**virtual ~base(); //** ជា destructor របស់ base class

**};**

**class derived: public base {**

**public :**

**virtual void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**virtual ~derived(); //** ជា destructor របស់ derived class

**};**

**void base::display(){**

**cout << "base class member function." << endl;**

**}**

**base::~base(){**

**cout << "base class destructor is called.\n";**

**}**

**void derived::display(){**

**cout << "derived class member function\n";**

**}**

**derived::~derived(){**

**cout << "derived class destructor is called.\n";**

**}**

**void main(){**

**base \*ptr = new derived;**

**ptr->display(); //** method ដែលមានឈ្មោះដូចគ្នាត្រូវបានហៅប្រើទាំងក្នុង

**//** base class ទាំងក្នុង derived class

**delete ptr; //** ការប្រើdelete operatorដើមី្បលុប ptr ពី memory

**}**

ឧទាហរណ៍ទី ៣ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើតនិងប្រកាស class មួយដែលមានទាំង virtual members និង virtual destructors នៅក្នុងលក្ខណៈ multiple inheritance។

**// virtual destructor function**

**#include <iostream.h>**

**class base {**

**public :**

**virtual void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ virtual

**virtual ~base(); //** ជា destructor របស់ base class

**};**

**class derived:public base {**

**public :**

**void display(); //**ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**~derived(); //** ជា destructor របស់ derived class

**};**

**void base::display(){**

**cout << "base class member function." << endl;**

**}**

**base::~base(){**

**cout << "base class destructor is called.\n";**

**}**

**void derived::display(){**

**cout << "derived class member function\n";**

**}**

**derived::~derived(){**

**cout << "derived class destructor is called.\n";**

**}**

**void main(){**

**base \*ptr = new derived;**

**ptr->display(); //** method ដែលមានឈ្មោះដូចគ្នាត្រូវបានហៅប្រើទាំងក្នុង

**//** base class ទាំងក្នុង derived class

**delete ptr; //** ការប្រើdelete operatorដើមី្បលុប ptr ពី memory

**}**

១១. អំពី Virtual Base class

យើងបានសិក្សារួចហើយនៅក្នុងជំពូកទី ៩ អំពី "Inheritance" ដែលនិយាយថា multiple inheritance គឺជាដំណើរការនៃការបង្កើត class ថ្មីមួយដោយទទួលលក្ខណៈពី base class លើសពីមួយ។ លំដាប់ថ្នាក់ multiple inheritance អាចមានលក្ខណៈសាំញ៉ាំដែលអាចនាំទៅដល់សភាពមួយ គឺ derived class ទទួលលក្ខណៈជាច្រើនដងពី base class មិនផ្ទាល់តែ មួយ។

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញនូវរបៀប base class ដែលត្រូវបានទទួល លក្ខណៈពីរដង ពី derived class តាមវិធីផ្សេងគ្នា។

**class baseA {**

**protected :**

**int x;**

**....**

**....**

**};**

**class derivedB:public baseA{ //** path 1តាមរយៈderivedB

**protected :**

**....**

**....**

**};**

**class derivedC:public baseA{ //** path 2តាមរយៈderivedC

**protected :**

**....**

**....**

**};**

**class abc:public derivedB, public derivedC {**

**// the data member x comes twice**

**....**

**....**

**};**

data member ឈ្មោះ x បានទទួលពីរដងនៅក្នុង derived class ឈ្មោះ abc ពោលគឺ លើកទីមួយតាមរយៈ derived class ឈ្មោះ derivedB និងម្តងទៀតតាមរយៈ derivedC។ លក្ខណៈនេះធ្វើឲ្យច្រឡំនិងគ្មានបានការឡើយ។ ភាពច្រំដែលជាច្រើនដងនៃ data member ខាងលើនេះអាចកែតម្រូវបានដោយធ្វើការផ្លាស់ប្តូរ derived class ឈ្មោះ derivedB និង derivedC ទៅជា virtual base class។ base class ណាក៏ដោយឲ្យតែវាប្រកាសដោយមានប្រើពាក្យ virtual នោះត្រូវបានហៅថា virtual base class។ Virtual base class គឺជាវិធីដ៏មានសារៈប្រយោជន៍ ដើម្បីជៀសវាងភាពច្រំដែលនៃ data member ដូចគ្នា ដែលមិនចាំបាច់នៅក្នុងលំដាប់ថ្នាក់ multiple inheritance។

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញនូវរបៀប base class មួយត្រូវបានទទួល លក្ខណៈពី derived class តែម្តងគត់តាមរយៈ virtual base class។

**class baseA {**

**protected :**

**int x;**

**....**

**....**

**};**

**class derivedB:public virtual baseA { //** path 1 តាមរយៈderivedB

**protected :**

**....**

**....**

**};**

**class derivedC:public virtual baseA { //** path 2តាមរយៈderivedC

**protected :**

**....**

**....**

**};**

**class abc : public derivedB, public derivedC {**

**// the data member x comes only once**

**....**

**....**

**};**

ដោយសារការធ្វើឲ្យ derivedB និង derivedC ទៅជា virtual base classes សម្រាប់ abc នោះ ការចម្លង data member x អាចប្រើបានតែម្តងគត់។

class មួយអាចជា base class ធម្មតា និងជា virtual base class ទៀតផង នៅក្នុងទម្រង់ single inheritance។

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើត derived class ដោយមាន virtual class objects និង non-virtual class objects។

**class baseA {**

**protected :**

**int x;**

**....**

**....**

**};**

**class derivedB:public virtual baseA { //** path 1 តាមរយៈderivedB

**protected :**

**....**

**....**

**};**

**class derivedC:public virtual baseA { //** path 2តាមរយៈderivedC

**protected :**

**....**

**....**

**};**

**class derivedD:public derivedB, public derivedC {**

**protected :**

**....**

**....**

**};**

**class derivedE:public baseA {**

**protected :**

**....**

**....**

**};**

**class abc:public derivedE, public derivedD {**

**....**

**....**

**};**

តាមការបង្ហាញពីខាងលើនេះ គេអាចសន្និដ្ឋានបានថា object នៃ derived class ឈ្មោះ abc នឹងមាន objects ពីរ របស់ baseA class គឺមួយមានលក្ខណៈជា virtual និងមួយទៀតមិនមានលក្ខណៈជា virtual ឡើយ។

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្កើតនូវ derived class ជាច្រើនដែលចូលទៅប្រើ data members នៃ base class ដូចគ្នាតាមរយៈ reference នៃ base class មិនផ្ទាល់។

**// using non-virtual base classes**

**#include <iostream.h>**

**class baseA {**

**protected :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ protected

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedB:public baseA { //** path 1 តាមរយៈderivedB

**​ protected :**

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ protected

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedC: public baseA { //** path 2តាមរយៈderived

**protected :**

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class abc:public derivedB, public derivedC {**

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void baseA::getdata(){**

**cout << "Enter an integer :\n";**

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**}**

**void baseA::display(){**

**cout << "Integer : " << x << endl;**

**}**

**void derivedB::getdata(){**

**baseA::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ baseA class

**cout << "Enter a floating point value :";**

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**baseA::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ baseA class

**cout << "Real number : " << y << endl;**

**}**

**void derivedC::getdata(){**

**baseA::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ baseA class

**cout << "Enter a string : ";**

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedC::display(){**

**baseA::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ baseA class

**cout << "String : " << name << endl;**

**}**

**void abc::getdata(){**

**derivedB::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ derivedB class

**derivedC::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ derivedC class

**}**

**void abc::display(){**

**derivedB::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ derivedB class

**derivedC::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ derivedC class

**}**

**void main(){**

**abc ob; //** ការបង្កើត object នៃ abc class

**ob.getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ object នៃ​ abc class

**ob.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ object នៃ​ abc class

**}**

data member ឈ្មោះ x ដែលមានលក្ខណៈជា protected ត្រូវបានចូលប្រើពីរដងដោយ derived classes ពីរ តាមរយៈ derivedB និង derivedD។ តម្លៃរបស់អញ្ញាតត្រូវបានចម្លងទៅក្នុងអញ្ញាត memory ពីរក្នុង heap។ វាប្រាកដណាស់ដែលមានភាពច្រំដែលៗកើតឡើងជាមួយនឹងទិន្នន័យផ្សេងគ្នា និងធ្វើឲ្យច្រឡំ ប្រសិនបើវាព្យាយាមដំណើរការនូវរបស់ (item) ដូចគ្នានៃ base class។

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្កើតនូវ derived class ជាច្រើនដែលចូលទៅប្រើ data members នៃ base class ដូចគ្នាតាមរយៈ reference នៃ base class មិនផ្ទាល់ដោយប្រើ virtual base class។

**// using virtual base classes**

**#include <iostream.h>**

**class baseA {**

**protected :**

**int x; //** ការប្រកាសអញ្ញាត x ជាប្រភេទ int មានលក្ខណៈ protected

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedB: public virtual baseA { //** path 1 តាមរយៈderivedB

**protected :**

**float y; //** ការប្រកាសអញ្ញាត y ជាប្រភេទ float មានលក្ខណៈ protected

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class derivedC: public virtual baseA {//** path 2 តាមរយៈderivedC

**protected :**

**char name[20]; //** ការប្រកាស name ជា array ប្រភេទ char មានប្រវែង 20 តួអក្សរ

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**class abc:public derivedB, public derivedC {**

**public :**

**void getdata(); //** ការប្រកាសអនុគមន៍ getdata() មានលក្ខណៈ public

**void display(); //** ការប្រកាសអនុគមន៍ display() មានលក្ខណៈ public

**};**

**void baseA :: getdata(){**

**cout << "Enter an integer:";//**បង្ហាញអក្សរ Enter an integer : មកលើអេក្រង់

**cin >> x; //** បញ្ចូលតម្លៃ x តាមរយៈ keyboard

**}**

**void baseA::display(){**

**cout << "Integer : " << x << endl;**

**}**

**void derivedB::getdata(){**

**baseA::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ baseA class

**cout << "Enter a floating point value :";**

**cin >> y; //** បញ្ចូលតម្លៃ y តាមរយៈ keyboard

**}**

**void derivedB::display(){**

**baseA::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ baseA class

**cout << "Real number : " << y << endl;**

**}**

**void derivedC::getdata(){**

**baseA::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ baseA class

**cout << "Enter a string :"; //** បង្ហាញអក្សរ Enter a string : មកលើអេក្រង់

**cin >> name; //** បញ្ចូលតម្លៃ name តាមរយៈ keyboard

**}**

**void derivedC::display(){**

**baseA::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ baseA class

**cout << "String : " << name << endl;**

**}**

**void abc::getdata(){**

**derivedB::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ derivedB class

**derivedC::getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ derivedC class

**}**

**void abc::display(){**

**derivedB::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ derivedB class

**derivedC::display(); //** ការហៅអនុគមន៍ display() តាមរយៈ derivedC class

**}**

**void main(){**

**abc ob; //** ការបង្កើត object នៃ abc class

**ob.getdata(); //** ការប្រកាសអនុគមន៍ getdata() តាមរយៈ object នៃ abc class

**ob.display(); //** ការប្រកាសអនុគមន៍ display() តាមរយៈ object នៃ abc class

**}**

ទោះជា data member ឈ្មោះ x ដែលមានលក្ខណៈជា protected ត្រូវបានចូលប្រើដោយ derived classes ពីរតាមរយៈ derivedB និង derivedD ក៏ដោយ តម្លៃរបស់អញ្ញាត x គឺដូចគ្នា។

សំណួរនិងលំហាត់

១ - អ្វីទៅលក្ខណៈ OOP? ចូរនិយាយពីលក្ខណៈ​ polymorphism និង​សារៈសំខាន់របស់វា។

២ - អ្វីទៅ Early binding? ចូរនិយាយពីគុណសម្បត្តិ និង គុណសម្បត្តិរបស់វា។

៣ - អ្វីទៅ Virtual function? ចូរឲ្យឧទាហរណ៍ជា code។

៤​ - តើគេប្រើ virtual function នៅពេលណា?

៥ - អ្វីទៅ Pure Virtual function? ចូរសរសេរ code បង្ហាញ។

៦ - អ្វីទៅ Late binding? ចូរនិយាយពីគុណសម្បត្តិ និង គុណសម្បត្តិរបស់វា។

៧ - ចូរប្រៀបធៀប Early binding និង Late binding។

៨ - អ្វីទៅ Abstract base class? ចូរឲ្យឧទាហរណ៍ជា code។

៩ - ចូរសរសេរ code បង្ហាញពីការប្រើ Constructor នៅក្នុង inheritance។

១០ - ចូរសរសេរ code បង្ហាញពីការប្រើ Destructor នៅក្នុង inheritance។

១១ - អ្វីទៅ Virtual Destructor? ចូរសរសេរ code បង្ហាញ។

១២ - អ្វីទៅ Virtual base class? ចូរឲ្យឧទាហរណ៍ជា code។ តើវាមានសារៈសំខាន់អ្វី?

១៣ - ចូរសរសេរ code បង្ហាញពីការប្រើលក្ខណៈ polymorphism ក្នុងភាសា C++។

១៤ - គេមាន Product class ដូចខាងក្រោម ៖

**class Product {**

**private:**

**int cod;**

**char name[20];**

**float price;**

**public:**

**virtual void input()=0;**

**virtual void output()=0;**

**};**

ចូរបង្កើត derived class ពីរចេញពី based class ខាងលើឈ្មោះ Product ដើមី្បឲ្យគេអាចយក

ទៅប្រើលក្ខណៈ polymorphism បាន។

១៥ - គេមាន class មួយដូចខាងក្រោម ៖

**class Shape {**

**public:**

**static float pi;**

**virtual float area()=0; //** គណនាក្រឡាផ្ទៃ

**virtual float perimeter()=0; //** គណនាបរិមាត្រ

**};**

ក- ចូរបង្កើត derived class ពីរចេញពី abstract based class ខាងលើឈ្មោះ Shape ដោយដាក់ឈ្មោះ Rectangle និង Circle។

ខ- ចូរសរសេរ code ដើមី្បបង្ហាញពីការប្រើលក្ខណៈ polymorphism បាន ក្នុង main() ដោយបង្កើតចំនួន 5 objects។