Table of Contents

មេរៀនទី៧: Classes and Objects

Chapter នេះសិក្សាលើមូលដ្ឋាននៃការដោះស្រាយបញ្ហាតាមបែប Object-Oriented Programming។ នៅក្នុង chapter នេះ រួមមានការសិក្សាទៅលើ:

- ការណែនាំអោយស្គាល់នូវលក្ខណៈទាំងឡាយនៃ Object-Oriented Programming (OOP)
- ការប្រកាស និងការបង្កើតចេញនូវ classes ដើម្បីបង្កបានជា object សំរាប់យកទៅអនុវត្តនៅក្នុងការដោះស្រាយបញ្ហាជាក់ស្តែង
- របៀបយក objects មកអនុវត្ត ដើម្បីអោយពួកវាធ្វើសកម្មភាពតាមរយៈ interface method ផ្សេងៗ
- បង្កើតចេញជា array នៃ objects ក្នុងការប្រើប្រាស់ទៅលើទំរង់ការងារដែលមានលក្ខណៈជា list
- ទំនាក់ទំនងរវាង pointers និង classes, និងការប្រើប្រាស់ pointers សំដៅទៅរក objects
- បង្កើតរបៀប ដោះស្រាយបញ្ហាមួយស្ថិតនៅក្នុងបញ្ហាមួយផ្សេងទៀត ដោយសំរេចឡើងតាមទំរង់ nested class
- ការបង្កើតចេញជា constructors របស់ class ដើម្បីអោយ objects របស់ class កើតមានឡើងភ្ជាប់ជាមួយនឹងតំលៃចាប់ផ្តើមសមស្របទៅតាមទំរង់ផ្សេងៗ
- ការបង្កើតបានជា dynamic object សំរាប់ឆ្លើយតបទៅនឹងតំរូវការប្រើប្រាស់ ទាំងឡាយនៅក្នុងពេល run
- ប្រភេទ function ដែលត្រូវអនុវត្តដោយស្វយ័ប្រវត្តិពេល object បានបាត់បង់ ត្រូវបានគេស្គាល់ថាជា destructor
- ការបង្កើតចេញ និងការប្រកាស static data ជាប្រភេទទិន្នន័យសាកលចំពោះបញ្ហាដែលត្រូវដោះស្រាយ
- friend functions, friend classes, និង common friend functions ដែលជាលក្ខណៈបន្ថែមទៅលើលក្ខណៈប្រតិបត្តិរបស់ OOP ក្នុងការបង្កើតទំនាក់ទំនងរវាង functions និង classes, classes និង classes

1. Introduction

Object នីមួយៗមាន attributes និង operations ផ្ទាល់ខ្លួនរបស់វា ហើយ objects ខ្លះ មានលក្ខណៈរស់រវើកជាង objects ផ្សេងទៀត។ គេអាចចាត់ objects ទាំងឡាយទៅក្នុងថ្នាក់មួយ ដូចជា objects បញ្ជាក់ពីម៉ូតូ ឡានតូច និងឡានធំជាដើម ត្រូវបានគេចាត់ទុកទៅ ក្នុងថ្នាក់យានយន្ត

(vehicle)។ ការបង្កើតកម្មវិធីដោយធ្វើការប្រើប្រាស់សញ្ញាណនៃ real-world objects ត្រូវបានគេហៅថា Object-Oriented Programming (OOP)។ ហេតុនេះការសិក្សាពី OOP គឺជាការសិក្សាអំពីដំណោះស្រាយដែលធ្វើការតំរែតំរង់ទៅរក objects។

នៅក្នុងមេរៀនទីមួយ គេបានធ្វើការបង្កើតកម្មវិធីដោយមិនបានទាញយកផលប្រយោជន៍នៃភាពខ្លាំងរបស់ C++ តាមរយៈការប្រើប្រាស់ classes និង objects នោះទេ។ ពាក្យថា class គឺជា keyword នៅក្នុង C++ ដែលត្រូវបានគេប្រើប្រាស់សំរាប់ច្របាច់បញ្ចូលគ្នានូវ data និង operations ទាំងឡាយទៅក្នុង entity មួយ។ ការបង្កើត class មួយឡើង នាំមកនូវលក្ខណៈនៃ data hiding, data abstraction, data encapsulation, inheritance (single, multiple), polymorphism និង method សំរាប់បង្កើតការទាក់ទងទៅនឹងសកម្មភាពណាមួយរបស់ object។

Data abstraction

នៅក្នុង OOP, abstraction ការត្រូវបានគេកំនត់ជាការប្រមូលផ្តុំនៃ data members និង function member (method) ទាំងឡាយ។

Data hiding

នៅក្នុង C++ ការបង្កើត class មួយឡើងអនុញ្ញាតិអោយមានការប្រកាស data member និង method ទាំងឡាយនៅក្នុងក្រុមនៃ public, private និង protected។ ភាពពិស្តានៃការប្រតិបត្តិ (implementation) របស់ class អាចត្រូវបានគេលាក់កំបាំង ដែលនេះគឺជាគោលការណ៍នៃ data hiding។

Data encapsulation

ទិន្នន័យខាងក្នុងរបស់ class ត្រូវបានគេផ្តាច់ចេញពីពិភពខាងក្រៅ ហើយដាក់វាស្របជាមួយនឹង method ទាំងឡាយនៅក្នុង capsule។ ការអនុវត្តដូចនេះ គឺដើម្បីជៀសវាងការប៉ះពាល់ដោយអចេតនាទៅដល់ទិន្នន័យ ពីពិភពខាងក្រៅ ក្នុងករណីដែលមានការប្រើប្រាស់មិនបានត្រឹមត្រូវទៅលើទិន្នន័យដែលមានសារៈសំខាន់។ Classes ទាំងឡាយរៀបចំចាត់ចែងដោយមានប្រសិទ្ធភាពទៅលើភាពស្មុគស្មាញនៃកម្មវិធីធំៗទាំងឡាយតាមរយៈ encapsulation។

Inheritance

C++ អនុញ្ញាតិអោយអ្នកសរសេរកម្មវិធីបង្កើតចេញនូវលំដាប់ថ្នាក់ (hierarchy) នៃ classes ទាំងឡាយ។ ការទទួលយកកេរដំណែលរបស់ classes ទាំងឡាយត្រូវបានគេប្រើប្រាស់សំរាប់បង្កើតចេញនូវលំដាប់ថ្នាក់នេះ។ លក្ខណៈពិសេសមូលដ្ឋានរបស់ base classes (parent classes) ទាំងឡាយអាចត្រូវបានគេបញ្ជូនទៅអោយ derived classes (child classes)។ នៅក្នុងការអនុវត្ត inheritance

បានកាត់បន្ថយនូវការសរសេរ code ដូចជាគេមិនចាំបាច់ធ្វើការសរសេរ base classes ឡើងវិញឡើយ ។

Polymorphism

នៅក្នុង OOP, Polymorphism ត្រូវបានកំនត់ឡើងនៅពេលដែលមានការអនុវត្តជំហានដំណើរការផ្សេងៗដោយការប្រើប្រាស់ method មួយអនុវត្ត message ដូចគ្នា (message គឺជាអ្វីមួយដែលកើតមានឡើងចំពោះ object មួយ)។

2. Class Declarations

Class មានលក្ខណៈប្រហាក់ប្រហែលនឹង structure នៅក្នុងភាសា C ដែរ ដោយវាមិនគ្រាន់តែមាន data members ទេ ថែមទាំងមាន member functions ទៀតផង។ Data members មិនត្រូវបានធ្វើការដោះស្រាយនៅផ្នែកខាងក្រៅនៃ member functions នោះទេ។ members របស់ class ដែលរួមមាន data និង function ត្រូវបានគេកំនត់ឡើងនៅក្នុង section មួយក្នុងចំនោម section ទាំងបីខាងក្រោម:

private

នៅក្នុង section នេះ data members ឬ member functions គ្រាន់តែអាច access ពី member functions និង friend functions របស់ class នោះតែប៉ុណ្ណោះ។ ពួកវាមិនអាច access នៅពិភពខាងក្រៅ class បានឡើយ។

protected

រាល់ members ទាំងឡាយដែលគេបានកំនត់នៅក្នុង section នេះគ្រាន់តែអាច access ពី member functions និង friend functions របស់ class ឬក៏ពី member functions និង friend functions របស់ derived classes ទាញបានពី class នោះតែប៉ុណ្ណោះ។

public

រាល់ members ទាំងឡាយដែលគេបានកំនត់នៅក្នុង section នេះត្រូវបានគេហៅថា public members។ ពួកវាអាច access ពីគ្រប់ function ទាំងអស់។ Public implementation operation ត្រូវបានគេហៅថា methods ឬ interface ទៅនឹងផ្នែកខាងក្រៅរបស់ class។ Member functions ណាក៏ដោយដែលអាចបញ្ជូន message ទៅកាន់ object ត្រូវបានគេហៅថា interface method។

ទំរង់ទូទៅនៃការប្រកាស class មួយត្រូវបានកំនត់ឡើងដោយ:

```
class ClassName
{
    private / protected / public:
```

```
            data members;
            implementation operations
            list of friend functions
    };
```

ឧទាហរណ៍ទី១៖ កំនត់ class Date តាងកាលបរិច្ឆេទជាក់លាក់ណាមួយរួមមាន ថ្ងៃ ខែ និងឆ្នាំ។

```
class Date {
        int day, month, year; // default is private section
    public:
        void read();
        void print();
};
```

នៅក្នុងការប្រកាស class ខាងលើ គេបានបង្កើតចេញនូវ class មួយឈ្មោះថា Date ដែលមាន private data members បីដូចជា day, month, year និងមាន public member functions ពីរដូចជា read(), print() តាមរយៈ class Date នេះគេអាចបង្កើត object របស់វាដូចការប្រកាសអញ្ញាតផ្សេងៗដែរ។

```
    Date obj;
```

obj គឺជា instance មួយរបស់ class Date ហើយវាមានទិន្នន័យពិតប្រាកដសម្រាប់ ថ្ងៃ, ខែ, ឆ្នាំ និងមាន implementation operation: read() សម្រាប់កំនត់កាលបរិច្ឆេទ និង print() សម្រាប់បង្ហាញនូវកាលបរិច្ឆេទ របស់ object ផ្ទាល់ខ្លួនវា។

ឧទាហរណ៍ទី២៖ កំនត់ class Student តាងអោយនិស្សិតម្នាក់រួមមាន លេខសំគាល់, អាយុ, ភេទ, កំពស់ និងទំងន់។

```
class Student {
    private:              // data members
        long id;
        int age;
        char sex;
        float height;
        float weight;
    public:               // member functions
        void read();
        void print();
};
```

ការកំនត់នេះបានបង្កើតចេញនូវ class មួយឈ្មោះថា Student ដែលមាន data members: id, age, sex, height, weight និងមាន member functions: read(), print()។ Object នីមួយៗ class Student សុទ្ធតែមានផ្ទុកនូវទិន្នន័យខាងក្នុងទាំងប្រាំ (id, age, sex, height, weight) ព្រមទាំងមាន methods: read(), print() ដូចៗគ្នា។

Function មួយដែលគេប្រកាសធ្វើជា member របស់ class ត្រូវបានគេហៅថា member functions ឬ method។ Methods ភាគច្រើនត្រូវបានគេផ្តល់អោយវានូវ លក្ខណៈ access ជា public ព្រោះពួកវាត្រូវបាន គេហៅយកមកអនុវត្តនៅក្រៅ class រួមទាំងនៅក្នុង function main() ឬនៅក្នុង function ណាមួយ ផ្សេងទៀត។ Member functions របស់ class មួយត្រូវបាន design ឡើងដើម្បីអនុវត្តយកទិន្នន័យរបស់ object មកប្រើប្រាស់ ហើយពួកវាធម្មតាអាចត្រូវបានគេបែងចែកចេញជា 3 ប្រភេទគឺ manager function, accessor function និង implementor function។

Manager functions

Manager functions ត្រូវបានគេប្រើប្រាស់ដើម្បីអនុវត្តផ្តើមចេញនូវតំលៃ និងលុប តំលៃរបស់ instance variable របស់ object នៃ class។ ឧទាហរណ៍ខ្លះៗសម្រាប់ manager functions មានដូចជា constructor និង destructor។

Accessor function

Accessor member functions គឺជា functions ទាំងឡាយណាដែលផ្តល់អោយ នូវពត៌មានស្តីអំពី current state របស់ object មួយ។ ឧទាហរណ៍សម្រាប់ accessor functions គឺជា const member functions ដែលភាគច្រើនជា methods ដែលមានឈ្មោះផ្តើមដោយពាក្យថា get នាំមុខ ជា ឧទាហរណ៍ getDay(), getAge() ជាដើម។

Implementor function

Implementor functions គឺជា functions ទាំងឡាយណាដែលធ្វើការបំរែបំរួល ទិន្នន័យរបស់ objects។ Functions ទាំងនេះត្រូវបានគេចាត់ទុកថាជា mulators ដែលមានតួនាទីក្នុងការកំនត់នូវ លក្ខណៈនៃ data hiding និង data encapsulation។ ដែលភាគច្រើនជា methods ដែលមានឈ្មោះផ្តើមដោយពាក្យថា set ដូចជា setDay(), setAge() ជាដើម។

3. Accessing Members of Objects

Member មួយរបស់ object នៃ class អាចត្រូវបានគេ refer ដូចគ្នានឹងការ refer ទៅកាន់ field មួយរបស់ structure ដែរ ដោយអនុវត្តតាមរយៈការប្រើប្រាស់ ប្រមាណវិធី dot(.) operator។

```
object.dataMember;
object.functionMember();
```

ឧទាហរណ៍ទី៣: Accessing methods សម្រាប់អនុវត្តសកម្មភាពរបស់ object។

```
class Sample {
    private:
        int data1;
        int data2;
```

```
    public:
        void setData(int one, int two);
        int getData1();
        int getData2();
        void print();
};
void main(){
    Sample obj;
    obj.setData(23, 5);   // accessing the method setData()
    obj.print();          // accessing the method print()
}
```

Statement: **obj.setData(23, 5);** អនុវត្តកំនត់តំលៃទៅអោយ object obj ដោយធ្វើ អោយ data1 មានតំលៃ 23 និង data2 មានតំលៃ 5។ ចំនែកឯ statement: **obj.print();** អនុវត្តបង្ហាញចេញពតិមានរបស់ object obj។

តាមឧទាហរណ៍ទាំងបីខាងលើ គេឃើញថា member functions នៅក្នុង class នីមួយៗដែលគេ ប្រើប្រាស់សម្រាប់ធ្វើសកម្មភាពទំនាក់ទំនងណាមួយទៅនឹង object របស់ classes នោះពុំទាន់បានបញ្ជាក់ប្រាប់ពីការងារជាក់លាក់ណាមួយរបស់ ពួកវានោះទេ។ ហេតុនេះហើយ ដើម្បីកំនត់ការងារជាក់លាក់ទៅអោយពួកវា គេចាំបាច់ត្រូវកំណត់ definitions របស់ពួកវាទាំងនោះ។ Definitions របស់ member functions អាចត្រូវបានកំនត់ឡើងនៅក្នុង class scope ឬនៅក្រៅ class scope។

ឧទាហរណ៍ទី៤: Definition របស់ member functions កំនត់ឡើងនៅក្នុង class

```
class Sample {
    private:
        int data1;
        int data2;
    public:
        void setData(int one, int two){
            data1 = one;
            data2 = two;
        }
        int getData1() { return data1; }
        int getData2() { return data2; }
        void read();
        void print();
};
```

នៅក្នុងការកំនត់ class Sample នេះ definitions របស់ member functions: setData(), getData1() និង getData2() ត្រូវបានកំនត់ឡើងនៅក្នុង scope ដោយក្នុងនោះ method setData() សម្រាប់ផ្លាស់ប្តូរទិន្នន័យរបស់ object តាមរយៈ arguments ទាំងពីររបស់វា method getData1 សម្រាប់ return តំលៃ

របស់ទិន្នន័យ data1 និង method getData2() សម្រាប់ return តំលៃរបស់ទិន្នន័យ data2។ ចំនែកឯ methods ពីរផ្សេងទៀតគឺ read() និង print() មិនទាន់ត្រូវបានគេកំនត់ definition អោយវានោះទេ។

ការកំនត់ definition របស់ member functions មួយនៅក្រៅ class ត្រូវមានទំរង់ដូចខាងក្រោម:

returnType className::functionName([prameters]){ ... }

ឧទាហរណ៍ទី៥: Definition របស់ member functions កំនត់ឡើងនៅក្រៅ class

```
class Sample {
    private:
        int data1;
        int data2;
    public:
        void setData(int one, int two){
            data1 = one;
            data2 = two;
        }
        int getData1() { return data1; }
        int getData2() { return data2; }
        void read();
        void print();
};

void Sample::read() {
    cout << "Input Data One: "; cin >> data1;
    cout << "Input Data Two: "; cin >> data2;
}

void Sample::print() {
    cout << "Data One: " << data1 << endl;
    cout << "Data Two: " << data2 << endl;
}
```

ការកំនត់ class Sample លើកនេះ definitions របស់ function: read() និង print() ត្រូវបានកំនត់ឡើងនៅក្រៅ class ដោយក្នុងនោះ method read() សម្រាប់បញ្ចូលទិន្នន័យពី keyboard ទៅអោយ data1 និង data2 របស់ object ហើយ method print() សម្រាប់បញ្ចេញទិន្នន័យរបស់ object របស់ data1 និង data2 ទៅកាន់ screen។

កម្មវិធីទី១: អនុវត្តពីការប្រើប្រាស់ object របស់ class

```
#include <iostream.h>
#include <conio.h>

class TwoNumbers {
    private:
        float first;
        float second;

    public:
```

```
        void assign(float a, float b){ first = a; second = b; }
        float getFirst() { return first; }
        float getSecond() { return second; }
        float sum() { return first + second; }
        float sub() { return first - second; }

        void read();
        void print();
};

void TwoNumbers::read(){
    cout << "First Number : "; cin >> first;
    cout << "Second Number: "; cin >> second;
}

void TwoNumbers::print(){
    cout << "First Number : " << first << endl;
    cout << "Second Number: " << second << endl;
    cout << "Sum          : " << sum() << endl;
    cout << "Sub          : " << sub() << endl;
}

void main() {
    cout.setf(ios::showpoint|ios::fixed); cout.precision(2);

    TwoNumbers obj;
    obj.assign(30, 23);
    cout << "Information of two numbers" << endl;
    obj.print();

    float f = obj.getFirst();
    float s = obj.getSecond();
    obj.assign(f*f, s*s);

    cout << "\nAfter making them squared" << endl;
    cout << "Information of two numbers" << endl;
    obj.print();

    cout << "Input two numbers" << endl;
    obj.read();

    cout << "Information of two numbers" << endl;
    obj.print();
}
```

4. Array of Objects

Array គឺជា user-defined data type ដែលគ្រប់ធាតុរបស់វាផ្ទុកនៅក្នុង memory locations បន្តគ្នារដឹក។ Array នៃ objects គឺជា array មួយដែលគ្រប់ធាតុរបស់វាជា objects របស់ class ជាក់លាក់ណាមួយ។ ការប្រកាស array នៃ objects ត្រូវអនុវត្តតាមទំរង់:

```
classNname arrayName[NumberOfElements];
```

ឧទាហរណ៍ទី៥: ប្រកាស array នៃ objects របស់ class Sample:

```
Sample object[5];
```

តាមការប្រកាសនេះ គេបាន objects គឺជា array ដែលមាន 5 ធាតុ ដោយធាតុនីមួយៗគឺជា objects របស់ class Sample។ ដូចនេះ ធាតុនីមួយៗរបស់ objects: objects[0], objects[1],…, objects[4] សុទ្ធតែមាន private data: data1, data2; និង member functions: setData(), getData1(), getData2(), read(), print() ដូចៗគ្នា។

កម្មវិធីទី២ កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ array of objects

```
// File: rectangle.h
// Class: Rectangle representing a rectangle

#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream.h>

class Rectangle {
    private:
        float width, length;

    public:
        void setLength(float lng){ length = lng; }
        void setWidth(float wid){ width = wid; }

        float getWidth(){ return width; }
        float getLength(){ return length; }
        float area(){ return width * length; }
        void read();
        void print();
        void println(){ print(); cout << endl; }
};

void Rectangle::read(){
    cout << "Width: "; cin >> width;
    cout << "Length: "; cin >> length;
}
void Rectangle::print(){
    cout << width << "\t" << length << "\t" << area();
}

#endif

/* ------------------------------------------------------------- */

// File: prog2_2.cpp, demonstrating of array of objects

#include <rectangle.h>

void main(){
```

```
    Rectangle recs[5];
    float wids[5] = {4.0, 5.0, 2.0, 7.0, 6.0};
    float lngs[5] = {7.0, 6.0, 5.0, 9.0, 8.0};

    for(int k=0; k<5; k++){
        recs[k].setWidth(wids[k]);
        recs[k].setLength(lngs[k]);
    }

    cout.setf(ios::showpoint|ios::fixed); cout.precision(2);

    cout << "All rectangle are: " << endl;
    cout << "N°\tWidth\tLength\tArea" << endl;
    for(int k=0; k<5; k++){
        cout << (k+1) << "\t";
        recs[k].println();
    }
}
```

កម្មវិធីទី៣ កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ array of objects។ នៅក្នុងកម្មវិធីនេះគេអនុវត្តបញ្ចូលបណ្តោយ និងទទឹងរបស់ចតុកោណកែងដែលជា object របស់ class Rectangle រួចហើយបង្ហាញចេញជាពត៌មាន (បណ្តោយ, ទទឹង និងក្រឡាផ្ទៃ) របស់ចតុកោណកែងទាំងអស់តាមលំដាប់កើននៃក្រឡាផ្ទៃ។ Class Rectangle យកមកអនុវត្តនៅក្នុងកម្មវិធីនេះ ដែលវាស្ថិតនៅក្នុង header file rectangle.h ដូចធ្លាប់បានយកមកប្រើប្រាស់នៅ ក្នុងកម្មវិធីទី២។

```
// File: prog2_3.cpp, demonstrating of array of objects
#include <rectangle.h>
#include <conio.h>

void sort(Rectangle[], int);

void main() {
    Rectangle recs[50];
    int n = 0;

    cout << "Input Rectangles: ";
    do {
        cout << endl << "Rectangle " << (n+1) << endl;
        recs[n++].read();
        if(n>=50) break;
        cout << endl << "Press ESC to stop" << endl;
    } while(getch()!=27);     // 27 is an ASCII code for ESC key.

    cout.setf(ios::fixed | ios::showpoint); cout.precision(2);

    cout << endl << "Display all rectangle before sorting " << endl;
    cout << "Width\tLength\tArea" << endl;
    for(int k=0; k<n; k++){
        recs[k].println();
    }
    getch();
```

```
    sort(recs, n);
    cout << endl << "Display all rectangle after sorted " << endl;
    cout << "Width\tLength\tArea" << endl;
    for(int k=0; k<n; k++){
        recs[k].println();
    }
    getch();
}

void sort(Rectangle recs[], int n) {
    Rectangle temp;
    for(int i=0; i<n-1; i++) {
        for(int j=i+1; j<n; j++) {
            if(recs[i].area() > recs[j].area()) {
            temp = recs[i];
            recs[i] = recs[j];
            recs[j] = temp;
            }
        }
    }
}
```

5. Constructors

Constructors គឺជា member functions ពិសេសដែលអនុវត្តសំរាប់កំណត់ទ្រង់តំលៃចាប់ផ្តើមរបស់ object មួយដោយស្វយ័ប្រវត្តិ។ Constructors មានឈ្មោះដូចនឹងឈ្មោះរបស់ class ហើយពុំមាន return type នោះទេ សូម្បីតែប្រភេទទិន្នន័យជា int ក៏ដោយ។ Class មួយអាចមាន constructors ច្រើនដោយមានភាព ខុសប្លែកពីគ្នាអាស្រយ័ទៅតាម ចំនួនប៉ារ៉ាម៉ែត្រ និងប្រភេទប៉ារ៉ាម៉ែត្រ ហើយពួកវាជាទូទៅមានលក្ខណៈ access ជា public ឬ protected។ ហើយវាត្រូវបានអនុវត្តនៅពេលដែល object ត្រូវបានបង្កើត។

ខាងក្រោមគឺជាទំរង់ទូទៅ prototype របស់ constructor:

ClassName([parameters]){ // Constructor's Body };

ឧទាហរណ៍ទី៦: បង្កើត class ដែលមាន constructors

```
class Sample {
    private:
        int data1;
        int data2;

    public:
        Sample();                       // default constructor
        Sample(int one, int two);       // constructor with parameters
        Sample(Sample& sam);            // copy constructor

        void setData(int one, int two);
        int getData1();
        int getData2();
```

```
        void read();
        void print();
};
```

នៅក្នុងឧទាហរណ៍នេះ class Sample មាន constructors ចំនួនបី: Sample(), Sample(Sample& sam) និង Sample(int one, int two)។ Constructors ទាំងបីមានលក្ខណៈខុសៗគ្នា ដោយវាត្រូវបានគេប្រើប្រាស់សំរាប់កំណត់នូវតំលៃចាប់ផ្តើមរបស់ object តាមទំរង់ផ្សេងៗគ្នា។ ដោយក្នុងនោះ object របស់ class Sample ដែលបានកើតឡើងតាម constructor ទី១ មិនត្រូវការ arguments ទេ, ហើយ object ដែលបានកើតឡើងតាម constructor ទី២ ត្រូវការ arguments ជាចំនួនគត់ពីរសំរាប់ធ្វើជាតំលៃចាប់ផ្តើមរបស់ object, ចំណែកឯការបង្កើតចេញនូវ object តាមរយៈ constructor ទី៣ ត្រូវការ argument ជា object មានរូចជាស្រេចណាមួយ ក្នុងគោលបំនងសំរាប់កំណត់នូវតំលៃចាប់ផ្តើមរបស់ object ថ្មីមួយដោយពឹងផ្អែកលើ object មានរូចជាស្រេច។

ទោះបីជា class មួយអាចមាន constructors ច្រើនយ៉ាងណាក៏ដោយ ក៏គេធ្វើការបែងចែកពួកវាចេញជា 3 ផ្នែកគឺ default constructor, parameterized constructor និង copy constructor។

Default constructors

Default constructor គឺជា constructor ដែលគេប្រើប្រាស់សំរាប់កំណត់នូវតំលៃចាប់ផ្តើមរបស់ object ថ្មីមួយ ដោយមិនចាំបាច់ធ្វើការបញ្ជូនទិន្នន័យពីកន្លែងបង្កើត object នោះទេ។ Default constructor គឺជា constructor ដែលពុំមាន parameters ឬមាន formal parameters ទាំងអស់ជា default parameters។

កម្មវិធីទី៤ កម្មវិធីបញ្ជាក់ពី class ដែលមានការប្រើប្រាស់ default constructor ដែលពុំមាន parameters ទេ

```
#include <iostream.h>

class Sample{
    private:
        int data1;
        int data2;

    public:
        Sample(){ data1=0; data2=0; }
        Sample(int one, int two);
        Sample(Sample& sam);

        void setData(int one, int two){ data1=one; data2=two; }
        int getData1(){ return data1; }
        int getData2(){ return data2; }
        void read(){
            cout << "Input data1: "; cin >> data1;
```

```
            cout << "Input data2: "; cin >> data2;
        }
        void print(){
            cout << "Data1: " << data1 << endl;
            cout << "Data2: " << data2 << endl;
        }
};

void main() {
    Sample obj;       // default constructor executes

    cout << "Initialization of obj" << endl;
    obj.print();

    obj.setData(300, 200);
    cout << endl << "Now Data of obj" << endl;
    obj.print();
}
```

កម្មវិធីទី៥ កម្មវិធីបញ្ជាក់ពី class ដែលមាន default constructor ត្រូវការប៉ារ៉ាម៉ែត្រទាំងអស់មានលក្ខណៈជា default parameter

```
#include <iostream.h>

class Sample {
    private:
        int data1;
        int data2;

    public:
        Sample(int one=0, int two=0){ data1=one; data2=two; }
        Sample(Sample& sam);

        void setData(int one, int two){ data1=one; data2=two; }
        int getData1(){ return data1; }
        int getData2(){ return data2; }

        void read(){
            cout << "Input data1: "; cin >> data1;
            cout << "Input data2: "; cin >> data2;
        }
        void print(){
            cout << "Data1: " << data1 << endl;
            cout << "Data2: " << data2 << endl;
        }
};

void main() {
    Sample obj1;              // initialized by default value (0, 0)
    Sample obj2(35, 50);      // initialized by value (35, 50)

    cout << "Initialization of obj1" << endl;
    obj1.print();

    cout << "Initialization of obj2" << endl;
```

```
    obj2.print();
}
```

Parameterized constructors

Parameterized constructor គឺជា constructor ដែលគេប្រើប្រាស់សំរាប់កំណត់នូវតំលៃចាប់ផ្តើមរបស់ object ថ្មីមួយ ដោយចាំបាច់ធ្វើការបញ្ជូនទិន្នន័យពីកន្លែងបង្កើត object នោះ។ Parameterized constructor គឺជា constructor ដែលមាន formal parameters យ៉ាងហោចណាស់មួយ ។

កម្មវិធីទី៦ កម្មវិធីបញ្ជាក់ពី class ដែលមានការប្រើប្រាស់ parameterized constructor

```
#include <iostream.h>

class Sample {
    private:
        int data1;
        int data2;

    public:
        Sample(){ data1=0; data2=0; }
        Sample(int one, int two){ data1=one; data2=two; }
        Sample(Sample& sam);

        void setData(int one, int two){ data1=one; data2=two; }
        int getData1(){ return data1; }
        int getData2(){ return data2; }

        void read(){
            cout << "Input data1: "; cin >> data1;
            cout << "Input data2: "; cin >> data2;
        }
        void print(){
            cout << "Data1: " << data1 << endl;
            cout << "Data2: " << data2 << endl;
        }
};

void main() {
    Sample obj1;              // initialized by default constructor
    Sample obj2(35, 50);      // initialized by parameterized constructor

    cout << "Initialization of obj1" << endl;
    obj1.print();

    cout << "Initialization of obj2" << endl;
    obj2.print();
}
```

Copy constructors

Copy constructor គឺជាប្រភេទ constructor ដែលគេប្រើប្រាស់សំរាប់កំណត់នូវតំលៃចាប់ផ្តើម

របស់ object ថ្មីមួយមកពី object មានរួចជាស្រេចណាមួយ។ Class មួយមាន copy constructor តែមួយគត់ ហេតុនេះ class ណាក៏ដោយ បើមាន copy constructor ត្រូវមានយ៉ាងហោចណាស់ constructor មួយផ្សេងទៀត។ ទំរង់ទូទៅរបស់ copy constructor គឺ:

`ClassName([const] ClassName& other){ // Constructor's Body }`

កម្មវិធីទី៧ កម្មវិធីបញ្ជាក់ពី class ដែលមាន copy constructor

```
#include <iostream.h>

class Sample {
    private:
        int data1;
        int data2;

    public:
        Sample(int one=0, int two=0){ data1=one; data2=two; }
        Sample(Sample& sam){ data1=sam.data1; data2=sam.data2; }

        void setData(int one, int two){ data1=one; data2=two; }
        int getData1(){ return data1; }
        int getData2(){ return data2; }

        void read(){
            cout << "Input data1: "; cin >> data1;
            cout << "Input data2: "; cin >> data2;
        }
        void print(){
            cout << "Data1: " << data1 << endl;
            cout << "Data2: " << data2 << endl;
        }
};
void main() {
    Sample obj1(99);              // initialized by default value (99, 0)
    Sample obj2(35, 50);          // initialized by value (35, 50)
    Sample obj3(obj1);            // initialized by value of obj1
    Sample obj4 = obj2;           // initialized by value of obj2
    Sample obj5 = Sample(5, 6);   // initialized by another object

    cout << endl << "Initialization of obj1" << endl;
    obj1.print();

    cout << endl << "Initialization of obj2" << endl;
    obj2.print();

    cout << endl << "Initialization of obj3" << endl;
    obj3.print();

    cout << endl << "Initialization of obj4" << endl;
    obj4.print();

    cout << endl << "Initialization of obj5" << endl;
    obj5.print();
```

```
    cout << endl << "Initialization of no named object" << endl;
    Sample(111, 222).print();
}
```

ដោយសារ constructors ត្រូវបានគេប្រើប្រាស់សំរាប់កំណត់នូវតំលៃចាប់ផ្តើមរបស់ object ដូច្នេះរឿងនេះត្រូវតែអនុវត្តឡើងមុនការ អនុវត្ត statements ទាំងឡាយណាដែលទាក់ទិនជាមួយនឹងទិន្នន័យរបស់ object។ ការកំណត់នូវតំលៃចាប់ផ្តើមរបស់ object បែបនេះត្រូវអនុវត្តមុន body របស់ constructor។ ដែលមានទំរង់ទូទៅដូចខាងក្រោម:

```
ClassName([parameters]):dataMember1(exp1)[,dataMember2(epx2),…]
{
    // Constructor's body
}
```

ឧទាហរណ៍ទី៧: ការកំណត់តំលៃចាប់ផ្តើមតំលៃរបស់ object មុនពេលអនុវត្ត statements របស់ constructors ។

```
class Sample{
    private:
        int data1;
        int data2;

    public:
        Sample() : data1(0), data2(0){
            // body of constructor
        }
        Sample(int one, int two) : data1(one), data2(two){
            // body of constructor
        }
        Sample(Sample& other) : data1(other.data1), data2(other.data2){
            // body of constructor
        }
};
```

កម្មវិធីទី៨ កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ constructor ទាំងឡាយរបស់ class Point ដែលតាងអោយចំនុចមួយនៅក្នុងប្លង់ ។

```
// File: point.h
// Class: Point representing a point in plan

#ifndef POINT_H
#define POINT_H

#include <iostream.h>
#include <math.h>

class Point{
    private:
        float x;
        float y;
```

```
    public:
        Point() : x(0.0), y(0.0){}
        Point(float x, float y) : x(x), y(y){}
        Point(const Point& point) : x(point.x), y(point.y){}

        void read() {
            cout << "x: "; cin >> x;
            cout << "y: "; cin >> y;
        }
        void print(){
            cout << "(" << x << ", " << y << ")" << endl;
        }
        int equal(Point& point){
            return x==point.x && return y==point.y;
        }
        float hasDistanceFrom(Point& point){
            float dx = x - point.x;
            float dy = y - point.y;
            return sqrt(dx*dx + dy*dy);
        }
};

#endif

/* ------------------------------------------------------------- */

// File: pro2_8.cpp testing of constructors

#include <point.h>

void main() {
    Point p1;                // p1 is (0.0, 0.0)
    Point p2(2.0, 6.0);      // p2 is (2.0, 6.0)
    Point p3(p2);            // p3 is the same as p2

    cout << "p1 is at ";
    p1.print();

    cout << "p2 is at ";
    p2.print();

    cout << "p3 is at ";
    p3.print();

    cout << endl;
    if(p1.equal(p2))
        cout << "p1 and p2 are the same.";
    else
        cout << "p1 is part " << p1.hasDistanceFrom(p2) << " from p2.";

    cout << endl;
    if(p2.equal(p3))
        cout << "p2 and p3 are the same.";
    else
        cout << "p2 is part " << p2.hasDistanceFrom(p3) << " from p3.";
```

```
}
```

ពេលដែលគេប្រកាស array នៃ objects នោះធាតុនីមួយៗរបស់វាគឺជា object ដែលត្រូវបានផ្តើមតំលៃដោយសារ default constructor។ ចំនែកឯ initialization របស់ array នៃ objects នោះធាតុនីមួយៗរបស់ array ត្រូវបានផ្តើមតម្លៃដោយ constructor ណាមួយដែលមិនមែនជា default constructor។

កម្មវិធីទី៩ កម្មវិធីបញ្ជាក់ពីការ initialization របស់ array នៃ objects។ Class Point ត្រូវបានយកមកអនុវត្តនៅក្នុងកម្មវិធីនេះ ដែលវាស្ថិតនៅក្នុង header file point.h ដូចធ្លាប់បានយកមកប្រើប្រាស់នៅក្នុងកម្មវិធីទី៨។

```
#include <point.h>

void main(){
    int k;

    // every element is initialized by default constructor
    Point pntd[4];

    // every element is initialized by parameterized constructor
    Point pntp[3] = { Point(1, 2), Point(4, 3), Point(3, 2) };

    cout << endl << "Initialization of array pntd" << endl;
    for(k=0; k<4; k++){
        cout << "pntd[" << k << "]: ";
        pntd[k].print();
    }

    cout << endl << "Initialization of array pntp" << endl;
    for(k=0; k<3; k++){
        cout << "pntp[" << k << "]: ";
        pntp[k].print();
    }

    float d1 = pntp[0].hasDistanceFrom(pntp[1]);
    float d2 = pntp[1].hasDistanceFrom(pntp[2]);
    float d3 = pntp[2].hasDistanceFrom(pntp[0]);

    if(d1>=d2+d3 || d2>=d1+d3 || d3>=d1+d2)
        cout << "These 3 points cannot make a triangle." << endl;
    else
        cout << "These 3 points can make a triangle." << endl;
}
```

6. Destructors

Destructor គឺជា member functions ពិសេសមួយដែលត្រូវបានអនុវត្តដោយស្វយ័ប្រវត្តិនៅពេលដែល object ត្រូវបានបាត់បង់។ Destructor មានឈ្មោះដូចនឹងឈ្មោះរបស់ class និងមាន

សញ្ញា ~ (tidle) នៅពីមុខ ហើយពុំមាន return type នោះទេ សូម្បីតែ void ក៏ដោយ។ Class មួយអាចមាន Destructor តែមួយគត់ ហើយវាមិនមានប៉ារ៉ាម៉ែត្រនោះទេ។ វាត្រូវបានប្រើប្រាស់សម្រាប់ភាពប្រាកដប្រជាថា រាល់អ្វីៗរបស់ object ត្រូវតែបាត់បង់ទៅវិញដោយស្វយ័ប្រវត្តិ នៅពេលដែល object ត្រូវបានបញ្ចប់ ជាពិសេសជាងនេះទៅទៀត គេច្រើនប្រើប្រាស់វាសម្រាប់ release នូវ memory space ទាំងឡាយណាដែលបានបង្កើតដោយ object។

ខាងក្រោមគឺជាទំរង់ទូទៅ prototype របស់ destructor:

```
~ClassName(){ // Destructor's Body }
```

កម្មវិធីទី១០ កម្មវិធីបញ្ជាក់ពីការអនុវត្តរបស់ destructor

```
// File: student.h
// Class: Student a class for testing destructor

#ifndef STUDENT_H
#define STUDENT_H

#include <iostream.h>

class Student{
    private:
        int id;
        char name[25];
        char gender[7];
        int age;
        char address[50];
        char handPhone[11];

    public:
        Student(){
            static int autoNumber = 1;
            id = autoNumber++;
        }
        ~Student(){  // destructor
            cout << this << " this alocation is free." << endl;
        }
        void insert(){
            cout << "Id         : " << id << endl;
            cout << "Name       : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(name, 25);
            cout << "Gender     : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(gender, 7);
            cout << "Age        : "; cin >> age;
            cout << "Address    : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(address, 50);
            cout << "Hand Phone: "; cin.clear();
            cin.seekg(0, ios::end); cin.get(handPhone, 11);
        }
        void display(){
            cout << id << "\t" << name << "\t" << gender << "\t" << age
```

```
                    << "\t" << address << "\t" << handPhone << endl;
        }
};

#endif

/* ------------------------------------------------------------ */

// File: pro2_10.cpp testing of destructor

#include <student.h>
#include <conio.h>

void main(){

    Student stu[10];
    int n = 0;

    cout << "Input Students, and Press ESC to stop";
    do {
        cout << endl << "Student" << (n+1) << endl;
        stu[n++].insert();
        if(n>=10) break;
    } while(getch()!=27);     // 27 is an ASCII code for ESC key.

    cout << endl << "Id\tName\t\tGender\tAge\tAddress\t\tHand Phone"
         << endl;
    for(int k=0; k<n; k++){
        stu[k].display();
    }
    getch();
}
```

7. Static Class Members

Static members គឺជា members របស់ class តែមិនមែនជា members របស់ object នោះទេ។ Static members ត្រូវបានគេបែងចែកជាពីរប្រភេទគឺ static data members និង static member functions។

Static data members

Static data members គឺជាទិន្នន័យ (data members) របស់ class ដែលមានលក្ខណៈរួមទៅគ្រប់ objects ទាំងអស់របស់ class ព្រោះថាវាបានកើតមានឡើងតែម្តងគត់ដោយមិនមានលក្ខណៈដូច non-static data members នោះទេ ដែលកើតមានឡើងនៅគ្រប់ objects ទាំងអស់ ហើយជាកម្មសិទ្ធរបស់ objects នីមួយៗផ្សេងៗគ្នា។ Static data members មិនមែនជា members របស់ objects នោះទេ។ ហេតុនេះ objects មិនអាច access តាមរយៈ objects បានដោយផ្ទាល់នោះទេ, គឺវាត្រូវ access តាមរយៈ Class Name ដោយប្រើប្រាស់ scope operator (::)។

Static data members គឺត្រូវបានបង្កើតរួចជាស្រេចមុនពេលដែល object របស់ class ត្រូវបានបង្កើតឡើង។ ផលប្រយោជន៍នៃការប្រើប្រាស់ static data members គឺដើម្បីប្រកាសនូវទិន្នន័យ global ដែលគេអាច update នៅក្នុងខណៈណាមួយនៃកម្មវិធី ហើយមានឥទ្ធិពលទៅគ្រប់ objects ទាំងអស់ក្រោយពីបានធ្វើការកែប្រែរួចហើយ (modified)។

Static data members មានលក្ខណៈមួយចំនួនដូចខាងក្រោម:

- Access rule នៃ static data members របស់ class នៅតែដូចគ្នាផងដែរជាមួយនឹង data members, ជាឧទាហរណ៍ បើសិនជា static data members ត្រូវបានប្រកាសឡើងនៅក្នុងផ្នែក private នោះវាអាច access បានតែនៅក្នុង class ខ្លួនឯងតែប៉ុណ្ណោះ។ ជាទូទៅគេច្រើនប្រកាសវាជា public static data member។
- នៅពេលណាដែល static data members ត្រូវបានប្រកាសឡើង និងមានតំលៃរួចជាស្រេចហើយ វានឹងត្រូវបានផ្តល់ (share) ទៅគ្រប់ objects ទាំងអស់របស់ class ពោលគឺវាត្រូវបានក្លាយជា global data member នៃគ្រប់ objects ទាំងអស់របស់ class។
- Static data member គឺត្រូវបានបង្កើតឡើងជាមួយនឹង keyword **static** ហើយត្រូវបានផ្តើមតំលៃមុននឹង control block របស់ function main() ត្រូវបានចាប់ផ្តើមអនុវត្ត។

  ខាងក្រោមគឺជាទំរង់ទូទៅ static data member:

  ```
  static DataType staticDataMember1 [, staticDataMember2, … ];
  ```

  ខាងក្រោមគឺជាការផ្តើមតំលៃរបស់ static data member ត្រូវធ្វើឡើងនៅខាងក្រៅ class ក្នុងទំរង់ទូទៅ

  ```
  DataType ClassName::staticDataMember1 [ = value ];
  ```

ឧទាហរណ៍ទី៨: បញ្ជាក់ពីការប្រកាស និងការកំណត់តំលៃចាប់ផ្តើមរបស់ static data member។

```
class Student{
     ......
     ......
     static int count;
     ......
     ......
};

int Student::count = 0;
```

នៅក្នុងការប្រកាសខាងលើនេះ count គឺជា static data member។ វាមិនមែនជា data member របស់ object ណាមួយ នៃ class Student នោះទេ។ count ត្រូវបានកំណត់តំលៃចាប់ផ្តើមដោយ 0។

កម្មវិធីទី១១ កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ static data member។ នៅក្នុង class Student គេបានបង្កើត static data member ធ្វើជាមធ្យោបាយសំរាប់ រាប់ចំនួន objects ដែលនៅមានជីវិត (still alive)។

```
#include <iostream.h>
#include <conio.h>

class Student{
    private:
        int id;
        char name[25];
        char gender[7];
        int age;
        char address[50];
        char handPhone[11];

    public:
        static int count;

        Student(){
            static int autoNumber = 1;
            id = autoNumber++;
            count++;
        }
        ~Student(){  // destructor
            cout << this << " this alocation is free." << endl;
            count--;
        }
        void insert(){
            cout << "Id        : " << id << endl;
            cout << "Name      : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(name, 25);
            cout << "Gender    : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(gender, 7);
            cout << "Age       : "; cin >> age;
            cout << "Address   : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(address, 50);
            cout << "Hand Phone: "; cin.clear();
            cin.seekg(0, ios::end); cin.get(handPhone, 11);
        }
        void display(){
            cout << id << "\t" << name << "\t" << gender << "\t" << age
                 << "\t" << address << "\t" << handPhone << endl;
        }
};

int Student::count = 0;

void main(){

    cout << "There are " << Student::count << " object alive" << endl;

    Student stu[10];
    int n = 0;
```

```
    cout << "Input Students, and Press ESC to stop";
    do {
        cout << endl << "Student" << (n+1) << endl;
        stu[n++].insert();
        if(n>=10) break;
    } while(getch()!=27);     // 27 is an ASCII code for ESC key.

    cout << "There are " << Student::count << " object alive" << endl;

    cout << endl << "Id\tName\t\tGender\tAge\tAddress\t\tHand Phone"
         << endl;
    for(int k=0; k<n; k++){
        stu[k].display();
    }
    getch();
}
```

Static member functions

Static member functions គឺជា member functions របស់ class ដែលគេប្រើប្រាស់សំរាប់អនុវត្ត ទៅលើ static data members។ Static member functions មិនមែនជា members របស់ objects នោះទេ។ ហេតុនេះ static member functions មិនអាច access តាមរយៈ objects បានដោយផ្ទាល់នោះទេ, គឺវាត្រូវ access តាមរយៈ Class Name ដោយប្រើប្រាស់ scope operator (::)។

ខាងក្រោមគឺជាទំរង់ទូទៅរបស់ static member functions

```
static returnType functionName([parameters]);
```

ឧទាហរណ៍ទី៩០: បញ្ជាក់ពីការប្រើប្រាស់នូវ static function member របស់ class។

```
class Student{
    ......
    ......
    static void tableHeader();
    ......
    ......
};
```

កម្មវិធីទី១២ កែប្រែ class និង function main() នៃកម្មវិធីទី១១ ដើម្បីបញ្ជាក់ពីការប្រើប្រាស់ static function member។

```
#include <iostream.h>
#include <conio.h>

class Student{
    private:
        int id;
        char name[25];
        char gender[7];
        int age;
        char address[50];
        char handPhone[11];
```

```
    public:
        Student(){
            static int autoNumber = 1;
            id = autoNumber++;
        }
        ~Student(){  // destructor
            cout << this << " this alocation is free." << endl;
        }

        void insert(){
            cout << "Id         : " << id << endl;
            cout << "Name       : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(name, 25);
            cout << "Gender     : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(gender, 7);
            cout << "Age        : "; cin >> age;
            cout << "Address    : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(address, 50);
            cout << "Hand Phone: "; cin.clear();
            cin.seekg(0, ios::end); cin.get(handPhone, 11);
        }
        static void tableHeader() {
            cout << "Id\tName\t\tGender\tAge\tAddress\t\tHand Phone"
                 << endl;
        }
        void display(){
            cout << id << "\t" << name << "\t" << gender << "\t" << age
                 << "\t" << address << "\t" << handPhone << endl;
        }
};
void main(){

    Student stu[10];
    int n = 0;

    cout << "Input Students, and Press ESC to stop";
    do {
        cout << endl << "Student" << (n+1) << endl;
        stu[n++].insert();
        if(n>=10) break;
    } while(getch()!=27);     // 27 is an ASCII code for ESC key.

    // Accessing static function member tableHeader
    Student::tableHeader();
    for(int k=0; k<n; k++){
        stu[k].display();
    }
    getch();
}
```

កម្មវិធីទី១៣ សម្រាប់ការអនុវត្តកម្មវិធី ស្តីពីការប្រើប្រាស់ class ។

```
#include <conio.h>
#include <iostream.h>
#include <time.h>
```

```
#include <string.h>
#include <stdlib.h>

int iSelection(int minY, int maxY, int x=30, int y=3);
void iDelay(float second);

const size = 100;
static int num = 0;

class Student {
    private:
        int id;
        char name[25], gender[7];
        int age;
        char address[50], phoneNumber[11];
        float khm, bio, his, phy, mat, total, average;
        char grade[10];
        int classified;

        void setTotal(){
            total = khm + bio + his + phy + mat;
        }
        void setAverage(){
            average = total / 5.0;
        }
        void setGrade(){
            if(average<50){
                strcpy(grade, "Fail");
            } else if(average<65){
                strcpy(grade, "Fair");
            } else if(average<75){
                strcpy(grade, "Good");
            } else if(average<85){
                strcpy(grade, "Very Good");
            } else {
                strcpy(grade, "Excellent");
            }
        }

    public:
        Student(){
            static int autoNumber = 1;
            id = autoNumber++;
        }
        int getId(){ return id; }

        ~Student(){
            cout << this << " this allocation is free." << endl;
        }
        void insert(){
            cout << "ID: " << id << endl;

            cout << "Name: "; cin.clear();
            cin.seekg(0, ios::end); cin.get(name, 25);
```

```
        cout << "Gender: "; cin.clear();
        cin.seekg(0, ios::end); cin.get(gender, 7);

        cout << "Age: "; cin >> age;

        cout << "Addresss: "; cin.clear();
        cin.seekg(0, ios::end); cin.get(address, 50);

        cout << "Phone Number: "; cin.clear();
        cin.seekg(0, ios::end); cin.get(phoneNumber, 11);

        cout << "Khmer: "; cin >> khm;
        cout << "Biology: "; cin >> bio;
        cout << "History: "; cin >> his;
        cout << "Physic: "; cin >> phy;
        cout << "Mathematic: "; cin >> mat;

        setTotal(); setAverage(); setGrade();
    }

    void setName(char name[25]){
        strcpy(this->name, name);
    }
    char* getName(){ return name; }

    void setGender(char gender[7]){
        strcpy(this->gender, gender);
    }
    char* getGender(){ return name; }

    void setAge(int age){ this->age = age; }
    int getAge(){ return age; }

    void setAddress(char address[50]){
        strcpy(this->address, address);
    }
    char* getAddress(){ return address; }

    void setPhoneNumber(char phoneNumber[11]){
        strcpy(this->phoneNumber, phoneNumber);
    }
    char* getPhoneNumber(){ return phoneNumber; }

    void setKhmer(float khm){
        this->khm = khm;
        setTotal(); setAverage(); setGrade();
    }
    float getKhmer(){ return khm; }

    void setBiology(float bio){
        this->bio = bio;
        setTotal(); setAverage(); setGrade();
    }
    float getBiology(){ return bio; }
```

```
        void setHistory(float his){
            this->his = his;
            setTotal(); setAverage(); setGrade();
        }
        float getHistory(){ return his; }

        void setPhysic(float phy){
            this->phy = phy;
            setTotal(); setAverage(); setGrade();
        }
        float getPhysic(){ return phy; }

        void setMathematic(float mat){
            this->mat = mat;
            setTotal(); setAverage(); setGrade();
        }
        float getMathematic(){ return mat; }

        float getTotal(){ return total; }
        float getAverage(){ return average; }
        char* getGrade(){ return grade; }

        static void setClassified(Student stu[], int num){
            int cls = 1;

            for(int i=0; i<num; i++) {
                for(int j=0; j<num; j++) {
                    if(stu[i].average < stu[j].average) {
                        cls++;
                    }
                }
                stu[i].classified = cls;
                cls = 1;
            }
        }
        int getClassified(){ return classified; }

        static void sort(Student [], int, int, int);

        static void tableHeader(){
            char horizantal = 6;
            cout << "ID\tName\t\tGender\tTotal\tAverage\tGrade\t"
                 << "Classified" << endl;
            for(int i=1; i<=79; i++){
                cout << horizantal;
            }
            cout << endl;
        }

        void display(){
            cout << id << "\t" << name << "\t" << gender << "\t" << total
                 << "\t" << average << "\t" << grade << "\t" << classified
                 << endl;
        }
```

```
        static void tableHeaderDetail(){
            char horizantal = 6;
            cout << "ID\tName\t\tGender\tKhmer\tBiology\tHistory\tPhysic"
                 << "\tMathematic\tTotal\tAverage\tGrade\tClassified"
                 << endl;
            for(int i=1; i<=79; i++){
                cout << horizantal;
            }
            cout << endl;
        }

        void displayDetail(){
            cout << id << "\t" << name << "\t" << gender << "\t"
                 << khm << "\t" << bio << "\t" << his << "\t"
                 << phy << "\t" << mat << "\t" << total << "\t"
                 << average << "\t" << grade << "\t" << classified
                 << endl;
        }
};

void Student::sort(Student stu[], int field, int num, int order=0){

    Student temp;

    // sort by id
    if(field==1){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].id > stu[j].id){
                        temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].id < stu[j].id){
                        temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by name
    else if(field==2){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                if(strcmp(stu[i].name, stu[j].name) > 0){
                    temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                if(strcmp(stu[i].name, stu[j].name) < 0){
                    temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
```

```
                }
        }
    }
    // sort by gender
    else if(field==3){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(strcmp(stu[i].gender, stu[j].gender) > 0){
                        temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(strcmp(stu[i].gender, stu[j].gender) < 0){
                        temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by age
    else if(field==4){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].age > stu[j].age){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].age < stu[j].age){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by address
    else if(field==5){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(strcmp(stu[i].address, stu[j].address) > 0){
                        temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(strcmp(stu[i].address, stu[j].address) < 0){
                        temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by total
```

```
    else if(field==6){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].total > stu[j].total){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].total < stu[j].total){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by average
    else if(field==7){
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].average > stu[j].average){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].average < stu[j].average){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by grade
    else if(field==8) {
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(strcmp(stu[i].grade, stu[j].grade) < 0){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
        else{
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
                    if(stu[i].grade < stu[j].grade){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
        }
    }
    // sort by classified
    else if(field==9) {
        if(order==0){
            for(int i=0; i<num-1; i++)
                for(int j=i+1; j<num; j++)
```

```
                    if(stu[i].classified > stu[j].classified){
                      temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                    }
          }
          else{
               for(int i=0; i<num-1; i++)
                    for(int j=i+1; j<num; j++)
                         if(stu[i].classified < stu[j].classified){
                           temp=stu[i]; stu[i]=stu[j]; stu[j]=temp;
                         }
          }
     }
}

void main(){
     Student stu[size];
     char ch;

     do{
          again: clrscr();
          gotoxy(8, 3);  cout << "  1. Insert          ";
          gotoxy(8, 4);  cout << "  2. Edit and Update ";
          gotoxy(8, 5);  cout << "  3. Delete          ";
          gotoxy(8, 6);  cout << "  4. Sort            ";
          gotoxy(8, 7);  cout << "  5. Search          ";
          gotoxy(8, 8);  cout << "  6. Display         ";
          gotoxy(8, 9);  cout << "  7. About Us        ";
          gotoxy(8, 10); cout << "  8. Exit            ";

          int choice = 0;
          choice = iSelection(3, 10);

          switch(choice){
               case 1:
                    clrscr();
                    stu[num].insert(); num++;
                    Student::setClassified(stu, num);
                    break;
               case 2:
                    // Edit and Update
                    break;
               case 3:
                    // Delete
                    break;
               case 4:
                    Student::sort(stu, 9, num);
                    goto again;
               case 5:
                    // Search
                    break;
               case 6: {
                    clrscr();
                    Student::tableHeader();
                    for(int i=0; i<num; i++){
                         stu[i].display();
```

```
                    }
                    break;
               }
               case 7:
                    // About Us
                    break;
               case 8:
                    exit(0);
               default:
                    cout << "Invalid Case!" << endl;
                    break;
          }
          gotoxy(2, 24); cout << "Do You Wanna Continue? [Yes/No]: ";
          ch = getche();
     } while(ch=='Y' || ch=='y');

     getch();
}

int iSelection(int minY, int maxY, int x, int y){
     char ch, symbol = 15;
     do{
          do{
               gotoxy(x, y); cout << symbol; iDelay(0.01);
          } while(!kbhit());

          ch = getch();
          if (ch=='\120'){ // down arrow
               gotoxy(x, y); cout<<"  "; y++;
          }
          if (ch=='\110') {// up arrow
               gotoxy(x, y); cout<<"  "; y--;
          }
          if(y>maxY){ y = minY; }
          if(y<minY){ y = maxY; }
     } while(ch!=13);

     return (y-(minY-1));
}
void iDelay(float second){
     clock_t endwait;
     endwait = clock() + second * CLOCKS_PER_SEC;
     do{
          //
     }while(!kbhit() && clock() < endwait);
}
```

មេរៀនទី ៨: Inheritances

នៅក្នុងមេរៀននេះយើងនឹងសិក្សាលើការបន្ត និងពង្រីកបន្ថែមការងាររបស់ class ដែលគេបាន design និងកំនត់ឡើងរួចរាល់ហើយ។ ទិដ្ឋភាពដ៏មានសារៈសំខាន់មួយទៀតនៃ OOP គឺការទទួលយកកេរមរតក (inheritance)។ Inheritance បង្ករអោយមានភាពងាយស្រួលក្នុងការពង្រីក class មួយ ឬក៏ផ្សំវាអោយទោជាការប្រើប្រាស់ថ្មី។ វាមានការទាក់ទិនជាមួយនឹងការបង្កើត class ថ្មីដែលពឹងអាស្រយ័ទៅលើ ឬក៏ទាញយកមកពី class ដែលមានរួចជាស្រេចណាមួយ។ ចំនុចសំខាន់ៗនៅក្នុង មេរៀននេះរួមមាន:

- ការបង្កើត class ថ្មីមួយទទួលមរតកមកពី class មានរួចជាស្រេចតែមួយគត់
- ស្ថានភាព access របស់ member នៅក្នុង class បង្កើតថ្មី ដែលទទួលបានមកពី class មានរួចជាស្រេច ទៅតាមប្រភេទផ្សេងៗគ្នានៃមធ្យោបាយទទួលមរកត
- ការបង្កើត class ថ្មីមួយទទួលមរតកមកពី classes មានរួចជាស្រេចជាច្រើន

1. What is inheritance?

Inheritance គឺជាដំណើរបង្កើតចេញនូវ class ថ្មីមួយដែលពឹងអាស្រយ័ទៅលើ class ដែលមានរួចជាស្រេច។ នៅក្នុងដំណើរនេះ class ដែលមានរួចជាស្រេចត្រូវបានគេហៅថា **base class** (ឬ parent class) ហើយ class បង្កើតថ្មីត្រូវបានគេហៅថា **derived class** (ឬ child class)។

Derived class ទទួលបាននូវគ្រប់សមត្ថភាព របស់ base class។ វាបានបន្ថែមនូវលក្ខណៈពិសេសថ្មីទៀតទៅអោយខ្លួនវាដោយពុំមានការប៉ះពាល់ ឬកែប្រែអ្វីៗរបស់ base class នោះទេ។ សារៈប្រយោជន៍ដែលទាញយកបានតាមរយៈ inheritance គឺ

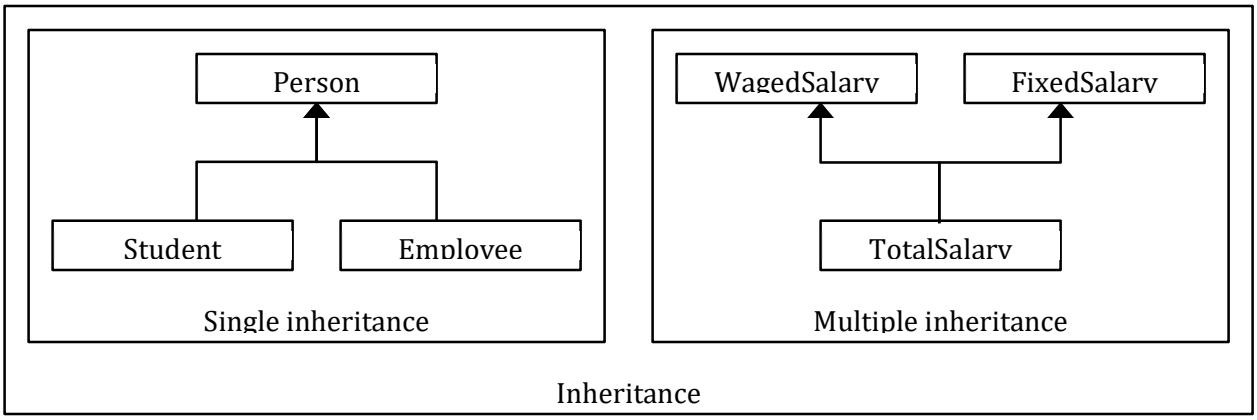
- ប្រើប្រាស់សមត្ថភាពរបស់ code ឡើងវិញដោយពុំបាច់ត្រូវសរសេរម្តងទៀត
- បង្កើនជំនឿចិត្តទៅលើ code ដោយមិនចាំបាច់ត្រូវបារម្ភអំពីកំហុស logic នោះទេ
- បន្ថែមលក្ខណៈប្រសើរៗ ថ្មីៗ ទៅលើ code

នៅក្នុង C++, inheritance ត្រូវបានបែងចែកជាពីរប្រភេទគឺ **Single** inheritance និង **Multiple** inheritance ។

- Single inheritance គឺជាការទទួលមរតកមកពី base class តែមួយ។ ជាឧទាហរណ៍ ទិន្នន័យរបស់នយោជិកម្នាក់ (employee) ទទួលកេរមរតកពីទិន្នន័យរបស់មនុស្សម្នាក់ (person) ព្រោះថានយោជិកម្នាក់ក៏ជាមនុស្សដែរ។
- Multiple inheritance គឺជាការទទួលមរតកមកពី base class ច្រើន។ ជាឧទាហរណ៍ ទិន្នន័យស្តី

ពីការទទួលប្រាក់ចំណូលគិតជាម៉ោង និងទទួលប្រាក់ចំណូលជាក់លាក់ រួមគ្នាទទួលកេរមរតកមកពីទិន្នន័យ ពីប្រាក់ចំណូលគិតជាម៉ោងផង និងប្រាក់ចំណូលជាក់លាក់ផង។

ខាងក្រោមគឺជារូបភាពបង្ហាញពី Single inheritance និង Multiple inheritance:



2. Single inheritance?

Single inheritance គឺជាការបង្កើតចេញនូវ class ថ្មីមួយដែលទទួលបាននូវគ្រប់សមត្ថភាពរបស់ class ដែលមានរួចជាស្រេច។

ទំរង់ទូទៅនៃការប្រកាស derived class តាម single inheritance គឺ:

```
class DerivedClassName : [Derivation] BaseClassName{
    // Members of Derived class (Specialized)
};
```

- BaseClassName គឺជាឈ្មោះរបស់ class ដែលមានរួចជាស្រេច
- DerivedClassName គឺជាឈ្មោះរបស់ derived class ដែលត្រូវទទួលមរតកពី base class
- Derivation គឺជាលក្ខណៈទទួលមរតករបស់ derived class មកពី base class។ វាអាចជា private (default), protected ឬជា public។

ឧទាហរណ៍ទី១o: បញ្ជាក់ពីការបង្កើតនូវទំរង់ inheritance តាមលក្ខណៈ single inheritance។

```
class Base {
    // Data members
    // Member functions
};

class Derived : public Base {
    // Additional data members
    // Additional member functions
};
```

នៅក្នុងឧទាហរណ៍ខាងលើ class ដែលមានឈ្មោះថា **Base** គឺជា base class ដែលត្រូវបានគេកំនត់រួចជាស្រេច ហើយវានឹងត្រូវបានផ្តល់នូវ members ទាំងអស់របស់វាទៅអោយ derived class រីង class

ដែលមានឈ្មោះថា **Derived** គឺជា derived class ដែលត្រូវបានទទួលមរតកពី base class ដែលវាត្រូវទទួលបាននូវគ្រប់លក្ខណៈទិន្នន័យ និងសមត្ថភាពទាំងអស់របស់ base class ដែលមាន។ រាល់ object ដែលបានបង្កើតចេញពី derived class នោះវានឹងមានគ្រប់លក្ខណៈទាំងអស់របស់ base class ផង និង derived class ផង។

កម្មវិធីទី១ សម្រាប់បញ្ជាក់ពីការប្រើប្រាស់នូវទំរង់ single inheritance ដោយក្នុងនោះ class Person គឺជា base class និង class Employee គឺជា derived class។

```
// Header File: person.h
// Class Person, representing a person

#ifndef PERSON_H
#define PERSON_H

#include <iostream.h>
#include <string.h>

class Person {
    protected:
        int id;
        char name[25];
        char gender[7];
        char address[50];
        char telephone[11];
        char email[50];

    public:
        Person(){
            static int autoNumber = 1;
            id = autoNumber++;
        }
        int getId(){ return this->id; }

        ~Person(){
            cout << this << " this allocation is free." << endl;
        }
        void insertPerson(){
            cout << "ID        : " << id << endl;

            cout << "Name      : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(name, 25);

            cout << "Gender    : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(gender, 7);

            cout << "Addresss  : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(address, 50);

            cout << "Telephone : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(telephone, 11);
```

```
            cout << "Email     : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(email, 50);
        }

        void setName(char name[25]){
            strcpy(this->name, name);
        }
        char* getName(){ return this->name; }

        void setGender(char gender[7]){
            strcpy(this->gender, gender);
        }
        char* getGender(){ return this->gender; }

        void setAddress(char address[50]){
            strcpy(this->address, address);
        }
        char* getAddress(){ return this->address; }

        void setTelephone(char telephone[11]){
            strcpy(this->telephone, telephone);
        }
        char* getTelephone(){ return this->telephone; }

        void setEmail(char email[50]){
            strcpy(this->email, email);
        }
        char* getEmail(){ return this->email; }
};

#endif

/* ------------------------------------------------------------- */

// Header File: employee.h
// Class Employee, representing a employee, inheriting from class Person

#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <person.h>

class Employee : public Person{
    private:
        char company[25];
        char department[25];
        float salary;

    public:
        void insertEmployee(){
            insertPerson();
            cout << "Company   : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(company, 25);

            cout << "Department: "; cin.clear();
```

```
            cin.seekg(0, ios::end); cin.get(department, 25);

            cout << "Salary    : "; cin >> salary;
        }

        void setCompany(char company[25]){
            strcpy(this->company, company);
        }
        char* getCompany (){ return this->company; }

        void setDepartment(char department[25]){
            strcpy(this->department, department);
        }
        char* getDepartment(){ return this->department; }

        void setSalary(float salary){
            this->salary = salary;
        }
        float getSalary(){ return this->salary; }

        static void tableHeader(){
            cout << "ID\tName\t\tGender\tTelephone\tDepartment\t\t"
                 << "Salary" << endl;
        }
        void display(){
            cout << id << "\t" << name << "\t" << gender << "\t"
                 << telephone << "\t" << department << "\t"
                 << salary << endl;
        }
};

#endif

/* ------------------------------------------------------------- */

/*
File: pro3_1.cpp, testing objects of class Employee, inherits from class
Person
*/

#include <employee.h>
#include <conio.h>

const size = 100;
static int num = 0;

void main(){

    Employee emp[size];
    char ch;
    clrscr();

    cout << "Input employees information: " << endl;
    do {
        cout << "Employee " << (num+1) << endl;
```

```
        emp[num++].insert();
        if(num>=size) break;

        cout << "Do you want to continiue? [Yes/No]: ";
        ch = getche();
    } while(ch=='Y'|| ch=='y');

    clrscr();
    Employee::tableHeader();
    for(int i=0; i<num; i++){
        emp[i].display();
    }
    getch();
}
```

3. Type of Derivation

នៅក្នុង inheritance, class ថ្មីដែលបានបង្កើតឡើងសំរាប់ការទទួលយកមរតកពី base class មានបីលក្ខណៈគឺ private, protected និង public។ តាមលក្ខណៈនៃ derivation នីមួយៗនេះវាបានធ្វើអោយ members នៃ derived class ដែលបានទាញមកពី base class មានស្ថានភាព access ផ្សេងៗគ្នាដូចមាននៅក្នុងតារាងខាងក្រោម:

Visibillity of Inheritance			
Visibillity of Base Class	**Visibillity of Derived Class**		
	private	**protected**	**public**
private	No Inheritance	No Inheritance	No Inheritance
protected	private	protected	protected
public	private	protected	public

តាមតារាងខាងលើនេះ យើងអាចធ្វើការកំនត់បានថា:

- គ្រប់ private members របស់ base class មិនអាច access នៅក្នុង derived class បានទោះជាត្រូវទទួលមរតកតាមលក្ខណៈ (derivation) បែបណាក៏ដោយ។
- ក្នុងករណីការទទួលមរតកតាមលក្ខណៈ private នោះបណ្តា members របស់ base class ទាំងអស់ ដែលមិនមែនជា private members វានឹងត្រូវក្លាយជា private members ទាំងអស់នៃ derived class។
- ក្នុងករណីការទទួលមរតកតាមលក្ខណៈ protected នោះបណ្តា members របស់ base class ទាំងអស់ ដែលមិនមែនជា private members វានឹងត្រូវក្លាយជា protected members ទាំងអស់នៃ derived class។
- ក្នុងករណីការទទួលមរតកតាមលក្ខណៈ public នោះបណ្តា members របស់ base class ទាំងអស់ ដែលមិនមែនជា private members វានឹងត្រូវក្លាយជា protected members នៃ derived class ប្រសិនបើវាជា protected members នៃ base class, ហើយវានឹងត្រូវក្លាយជា public

members នៃ derived class ប្រសិនបើវាជា public members នៃ base class។

ឧទាហរណ៍ទី២0: class Cylinder ទទួលមរតកមកពី class Circle តាមលក្ខណៈដូចខាងក្រោម:

```
class Circle {
    private:
        float radius;

    protected:
    float area();
    float perimeter();

    public:
    Circle(float radius = 1.0);
    Circle(Circle& circle);
    void read();
    void print();
    void println();
};
class Cylinder : protected Circle {
    private:
        float height;

    public:
        Cylinder(float radius = 1.0, float height = 1.0);
        Cylinder(Cylinder& cylinder);
        void read();
        void print;
        float area();
        float volume();
};
```

តាមការប្រកាសខាងលើ នោះស្ថានភាព access របស់ members ដែលបានទទួលពី class Circle វានឹងក្លាយទៅជា members របស់ class Cylinder មានលក្ខណៈដូចខាងក្រោម:

- Data member **radius** មានស្ថានភាព access ជា inaccessible
- Constructors ទាំងពីរ មានស្ថានភាព access ជា protected
- Member functions: area(), perimeter() មានស្ថានភាព access ជា protected
- Member functions: read(), print(), println() មានស្ថានភាព access ជា protected

3. Multiple inheritance?

Multiple inheritance គឺជាដំណើរនៃការបង្កើតចេញនូវ derived class ថ្មីមួយដែលទទួលមរតកមកពី base class ច្រើនផ្សេងគ្នា។ នៅក្នុងដំនើរនេះ derived class ទទួលបានគ្រប់សមត្ថភាពទាំងអស់របស់ base classes ទាំងអស់ និងរួមផ្សំជាមួយនឹងលក្ខណៈផ្ទាល់ខ្លួនបន្ថែមទៀត។ Rule នៃការទទួលមរតក និងការ access សំរាប់ multiple inheritance ដូចគ្នាទៅនឹង single inheritance ដែរ។

ទំរង់ទូទៅនៃការប្រកាស derived class តាម multiple inheritance គឺ:

```
class DerivedClassName : Derivation BaseClassName1
    , Derivation BaseClassName2 [, Derivation BaseClassName3, …]
{
    // Members of Derived class (Specialized)
};
```

កម្មវិធីទី២ សម្រាប់បញ្ជាក់ពីការប្រើប្រាស់នូវទំរង់ multiple inheritance ដោយក្នុងនោះ class Area, class Perimeter គឺជា base class និង class Rectangle គឺជា derived class។

```
/*
File: pro3_2.cpp, testing objects of class Rectangle, inherits from class
Area and class Perimeter
*/

#include <iostream.h>
#include <conio.h>
#include <math.h>

#define PI 3.14159265

class Area{
    public:
        float areaOfCircle(float radius){
            return PI * pow(radius, 2);
        }
        float areaOfEllipse(float majorAxis, float minorAxis){
            return PI * majorAxis * minorAxis;
        }
        float areaOfParallelogram(float width, float length){
            return width * length;
        }
        float areaOfRectangle(float width, float height){
            return width * height;
        }
        float areaOfSector(float radius, int angle){
            if(angle>0){
                if(angle%360==0){
                    angle = 360;
                } else {
                    angle %= 360;
                }
            }
            return angle/360 * PI * pow(radius, 2);
        }
        float areaOfSquare(float width){
            return pow(width, 2);
        }
        float areaOfTrapezoid(float smallBase, float bigBase,
            float height){
            return (small + bigBase) * height /2.0;
        }
        float areaOfTriangle(float base, float height){
```

```
            return (base * height / 2.0);
        }
};

class Perimeter{

    public:
        float perimeterOfCircle(float radius){
            return 2 * PI * radius;
        }
        float perimeterOfEllipse(float majorAxis, float minorAxis){
            return 2*PI*sqrt((pow(majorAxis, 2)*pow(minorAxis, 2))/2.0);
        }
        float perimeterOfParallelogram(float width, float length){
            return (width + length) * 2.0;
        }
        float perimeterOfRectangle(float width, float height){
            return (width + height) * 2.0;
        }
        float perimeterOfSector(float radius, int angle){
            if(angle>0){
                if(angle%360==0){
                    angle = 360;
                } else {
                    angle %= 360;
                }
            }
            return angle/360 * 2 * PI * radius;
        }
        float perimeterOfSquare(float width){
            return width * 4;
        }
        float perimeterOfTrapezoid(float firstLength, float secondLength,
            float thirdLength, float fourLength){
            return firstLength + secondLength + thirdLength + fourLength;
        }
        float perimeterOfTriangle(float firstLength, float secondLength,
            float thirdLength){
            return firstLength + secondLength + thirdLength;
        }
};

// Rectangle class is derived from classes Area and Perimeter.
class Rectangle : private Area, private Perimeter{

    private:
        float width, height;

    public:
        Rectangle() : width(0.0), height(0.0){}
        void setData(float width, float height) {
            this->width = width;
            this->height = height;
        }
        void readData(){
```

```
            cout << "Enter width: "; cin >> width;
            cout << "Enter height: "; cin >> height;
        }
        float area(){
            return areaOfRectangle(width, height);
        }
        float perimeter(){
            return perimeterOfRectangle(width, height);
        }
};

void main(){
    clrscr();
    Rectangle rec;

    cout.setf(ios::showpoint|ios::fixed); cout.precision(2);

    rec.readData();
    cout << "Area Of Rectangle is " << rec.area() << endl;
    cout << "Perimeter Of Rectangle is " << rec.perimeter() << endl;
    getch();
}
```

កម្មវិធីទី៣ សម្រាប់បញ្ជាក់ពីការប្រើប្រាស់នូវទំរង់ multiple inheritance ដោយក្នុងនោះ class Person ដោយទាញយកពី Header File: Person.h, class Employee គឺជា base class និង class Manager គឺជា derived class។

```
// Header File: employee.h
// Class Employee, representing a employee

#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <iostream.h>

class Employee{
    private:
        char company[25];
        char department[25];
        float salary;

    public:
        void insertEmployee(){
            cout << "Company   : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(company, 25);

            cout << "Department: "; cin.clear();
            cin.seekg(0, ios::end); cin.get(department, 25);

            cout << "Salary    : "; cin >> salary;
        }
        void setCompany(char company[25]){
            strcpy(this->company, company);
        }
```

```
        char* getCompany (){ return this->company; }

        void setDepartment(char department[25]){
            strcpy(this->department, department);
        }
        char* getDepartment(){ return this->department; }

        void setSalary(float salary){
            this->salary = salary;
        }
        float getSalary(){ return this->salary; }
};

#endif

/* ------------------------------------------------------------------ */

/*
Header File: manager.h
Class Manager, representing a manager, inheriting from class Person and class
Employee
*/

#ifndef MANAGER_H
#define MANAGER_H

#include <person.h>
#include <employee.h>

class Manager : public Person, public Employee{
    protected:
        char jobTitle[25];

    public:
        void insertEmployee(){
            insertPerson();
            insertEmployee();
            cout << "Job Title : "; cin.clear();
            cin.seekg(0, ios::end); cin.get(jobTitle, 25);
        }
        void setJobTitle(char jobTitle[25]){
            strcpy(this->jobTitle, jobTitle);
        }
        char* getJobTitle(){ return this->jobTitle; }

        static void tableHeader(){
            cout << "ID\tName\t\tGender\tTelephone\tDepartment\t\t"
                 << "Salary\tJob Title" << endl;
        }
        void display(){
            cout << id << "\t" << name << "\t" << gender << "\t"
                 << telephone << "\t" << department << "\t"
                 << salary << "\t" << jobTitle << endl;
        }
};
```

```
#endif

/* ------------------------------------------------------------ */

//File: pro3_3.cpp, testing objects of class Employee, inherits from Person

#include <manager.h>
#include <conio.h>

void main(){

    Manager man[size];
    char ch;
    clrscr();

    cout << "Input employees information: " << endl;
    do {
        cout << "Employee " << (num+1) << endl;
        man[num++].insert();
        if(num>=size) break;

        cout << "Do you want to continiue? [Yes/No]: "; ch = getche();
    } while(ch=='Y'|| ch=='y');

    clrscr();
    Manager::tableHeader();
    for(int i=0; i<num; i++){
        man[i].display();
    }
    getch();
}
```

មេរៀនទី ៩: Overloading

នៅក្នុងមេរៀននេះ យើងនឹងសិក្សាពីលក្ខណៈបំពេញបន្ថែមទៅលើការសរសេរកម្មវិធី និងលក្ខណៈអនុវត្តជាមួយនឹង objects របស់ class។ នៅក្នុងភាសា C++ ផ្តល់អោយយើងនូវការសរសេរ functions ទាំងឡាយដែលមានឈ្មោះដូចគ្នា និងប្រមាណវិធីជាច្រើន ដែលអនុញ្ញាតិអោយគេបង្កើតការងារសំរាប់អនុវត្តជាមួយនឹង objects។ ចំនុចសំខាន់ៗនៅក្នុង មេរៀននេះ រួមមាន:

- ការបង្កើត functions ជាច្រើនមានឈ្មោះដូចគ្នា តែការហៅយកមកអនុវត្តផ្សេងគ្នា ត្រូវបានគេហៅថា function overloading។
- ការបង្កើតការងារបន្ថែមរបស់ប្រមាណវិធីខ្លះៗ ដែលមានរួចជាស្រេចនៅក្នុងភាសា C++ ដូចជាប្រមាណវិធី +, >, +=, ... ជាដើម សំរាប់បង្កើនលទ្ធភាពអនុវត្តរបស់ objects នៅក្នុងកន្សោម ត្រូវបានគេហៅថា operator overloading។

1. What is function overloading?

Function overloading គឺជាដំណើរបង្កើតចេញនូវ functions ជាច្រើនដែល functions ទាំងអស់នោះមានឈ្មោះដូចគ្នា ប៉ុន្តែវាខុសគ្នាដោយ ប្រភេទទិន្នន័យរបស់អនុគមន៍ (function's return type), វាខុសគ្នាដោយ ចំនួននៃ parameters ឬខុសគ្នាដោយប្រភេទទិន្នន័យរបស់ប៉ារ៉ាម៉ែត្រ (parameters's data type) ហើយ functions ទាំងនោះត្រូវតែស្ថិតនៅក្នុង scope តែមួយ។

ឧទាហរណ៍ទី១០: ការបង្កើត functions overloading សំរាប់ធ្វើការផ្ទេរនូវតំលៃពីរ ពីគ្នាទៅវិញទៅមក ដែលយើងតាងឈ្មោះថា swap។

- Function ទី១មានឈ្មោះថា swap សំរាប់ធ្វើការផ្ទេរតំលៃទៅវិញទៅមក រវាងអញ្ញតិពីរ ប្រភេទជាចំនួនគត់។

```
void swap(int &x, int &y){
     int temp;
     temp = x;
     x = y;
     y = temp;
}
```

- Function ទី២មានឈ្មោះថា swap សំរាប់ធ្វើការផ្ទេរតំលៃទៅវិញទៅមក រវាងអញ្ញតិពីរ ប្រភេទជាចំនួនទសភាគ។

```
void swap(float &x, float &y){
     float temp;
     temp = x;
     x = y;
     y = temp;
```

```
}
```

- Function ទី៣មានឈ្មោះថា swap សំរាប់ធ្វើការផ្ទេរតំលៃទៅវិញទៅមក រវាងអញ្ញតិពីរ ប្រភេទជា string។

```
void swap(char str1[50], char* str[50]){
    char temp[50];
    strcpy(temp, str1);
    strcpy(str1, str2);
    strcpy(str2, temp);
}
```

តាមរយៈការកំនត់ចេញនូវ functions swap ទាំងបីខាងលើគេថាពួកវាគឺមានលក្ខណៈ overloading។ ហេតុនេះ ការហៅ function មកអនុវត្តផ្សេងគ្នាទៅតាម arguments ដែលពួកវាត្រូវការ

- ប្រសិនបើយើងហៅ swap function ដោយផ្តល់តំលៃទៅអោយ arguments របស់វាជាចំនួនគត់ទាំងពីរ នោះវានឹងហៅនូវ function swap ទី១។
- ប្រសិនបើយើងហៅ swap function ដោយផ្តល់តំលៃទៅអោយ arguments របស់វាជាចំនួនទសភាគទាំងពីរ នោះវានឹងហៅនូវ function swap ទី២។
- ប្រសិនបើយើងហៅ swap function ដោយផ្តល់តំលៃទៅអោយ arguments របស់វាជា string ទាំងពីរ នោះវានឹងហៅនូវ function swap ទី៣។

កម្មវិធីទី១ សម្រាប់បញ្ជាក់ពីការប្រើប្រាស់នូវ function overloading នៅក្នុង scope របស់ class មួយ

```
// Header File: rectangle.h
// Class Rectangle with overloading method, representing a rectangle

#ifndef RECTANGLE_H
#define RECTANGLE_H

#include <iostream.h>

class Rectangle{
    protected:
        float width;
        foat height;

    public:
        Rectangle();
        Rectangle(float width, float height);
        Rectangle(Rectangle&);

        void setData(float width, float height);
        void setData(Rectangle&);

        void read();
        void print();
        void println();
```

```
        float area();
        float perimeter();
        static void tableHeader();
};

Rectangle::Rectangle() : width(0), height(0){}

Rectangle::Rectangle(float width, float height)
    : width(width), height(height){}
Rectangle::Rectangle(Rectangle& rec)
    : width(rec.width), height(rec.height){}

void Rectangle::setData(float width, float height){
    this->width = width;
    this->height = height;
}
void Rectangle::setData(Rectangle& rec){
    this->width = rec.width;
    this->height = rec.height;
}
void Rectangle::read(){
    cout << "Width: "; cin >> width;
    cout << "Height: "; cin >> height;
}
void Rectangle::print(){
    cout.setf(ios::showpoint | ios::fixed); cout.precision(2);
    cout << width << "\t" << height << "\t" << perimeter()
         << "\t\t" << area();
}
void Rectangle::println(){ print(); cout << endl; }

float Rectangle::area(){ return width * height; }

float Rectangle::perimeter(){ return (width + height) * 2; }

void Rectangle::tableHeader(){
    cout << "Width\tHeight\tPerimeter\tArea" << endl;
}

#endif

/* ------------------------------------------------------------- */

/*
File: prog4_1.cpp, testing objects of class Rectangle with overloading
method.
*/

#include <rectangle.h>

void main(){
    Rectangle rec1;           // initialized by default constructor
    Rectangle rec2(4.5, 5.5); // initialized by parameterized constructor
    Rectangle rec3(rec1);     // initialized by copy constructor
```

```
    cout << "Initialization" << endl;

    cout << "Rectangles:" << "\t";
    Rectangle::tableHeader();
    cout << "Rectangle 1" << "\t"; rec1.println();
    cout << "Rectangle 2" << "\t"; rec2.println();
    cout << "Rectangle 3" << "\t"; rec3.println();

    // set rec1 by invoking method setData(Rectangle&)
    rec1.setData(rec3);

    // set rec2 by invoking method setData(float, float)
    rec2.setData(3.0, 5.5);

    // set rec3 by invoking method setData(float, float)
    rec3.setData(4.5, 5.2);

    cout << endl << "After setting" << endl;
    cout << "Rectangles:" << "\t";
    Rectangle::tableHeader();
    cout << "Rectangle 1" << "\t"; rec1.println();
    cout << "Rectangle 2" << "\t"; rec2.println();
    cout << "Rectangle 3" << "\t"; rec3.println();
}
```

2. Operator Overloading

Operator overloading គឺជាដំណើរបង្កើតចេញនូវការងារថ្មីរបស់ប្រមាណវិធីខ្លះៗដែលមានរួចជាស្រេចនៅក្នុង C++, ដែលសំរាប់ប្រើប្រាស់ជាមួយនឹង objects។ Operator overloading អាចអនុវត្តតាមមធ្យោបាយជា member functions ឬ non-member functions ក្នុងទំរង់ទូទៅដូចខាងក្រោម:

```
ReturnType operator OperatorSign ([parameters]);
```

ក្នុងនោះ

- ReturnType គឺជាប្រភេទទិន្នន័យរបស់កន្សោមប្រមាណវិធីដែលបាន overload
- operator គឺជា keyword ដែលយើងប្រើប្រាស់សំរាប់បញ្ជាក់ថាវាគឺជា operator overloading
- OperatorSign គឺជាប្រមាណវិធីមានរួចជាស្រេចនៅក្នុង C++ ហើយយើងអាច overload បាន
- parameter គឺជា list នៃប៉ារ៉ាមែត្រមានទំរង់ដូចប៉ារ៉ាមែត្ររបស់ function តែពុំមានលក្ខណៈជា default parameters ទេ។ ពួកវាដើរតួជាអង្គប្រមាណវិធីរបស់កន្សោម។

ឧទាហរណ៍ទី២០: ការប្រកាសប្រមាណវិធីសំរាប់យកទៅអនុវត្តជាមួយនឹង objects របស់ class Fraction

```
complex operator + (const complex& x, const complex& y);
```

នេះជាការប្រកាសប្រមាណវិធី សំរាប់អនុវត្តន៍ប្រមាណវិធីបូករបស់ objects ពីរ។

3. Assignment Operator Overloading

Assignment operator (=) អាច overloaded បានតាមមធ្យោបាយតែមួយគត់ គឺធ្វើជា function member របស់ class តែប៉ុណ្ណោះ ដោយវាត្រូវការប៉ារ៉ាម៉ែត្រតែមួយគត់ ដែលមានប្រភេទទិន្នន័យជាអ្វីក៏បាន។ ដែល assignment operator វាមានលក្ខណៈដូចនឹង copy constructor ដែរ។

ខាងក្រោមគឺជាទំរង់ទូទៅនៃការប្រកាស overloading ទៅលើ assignment operator គឺ:

```
class ClassName{
    ……
    ReturnType operator = (oneParameter);
    ……
};
```

ឧទាហរណ៍ទី៣: ការបង្កើត overloading នៃប្រមាណវិធី assignment operator នៅក្នុង class iArray សំរាប់ assign object មួយនៃ class iArray ទៅអោយ object មួយទៀតនៃ class តែមួយ។

```
class iArray{
    private:
        int *index;
        int size;

    public:
        iArray(int size=5);
        ~iArray();
        ……
        void operator = (iArray& iArr);
        ……
};
```

ក្នុងឧទាហរណ៍នេះ បានប្រកាសនូវ void operator = (iArray& iArr); ចេញជា overloading ប្រមាណវិធី assignment សំរាប់អនុវត្តជាកន្សោមក្នុងការផ្ទេរទិន្នន័យរបស់ object មួយទៅអោយ object មួយទៀតរបស់ class iArray ដូចគ្នា។

កម្មវិធីទី២ សម្រាប់បញ្ជាក់ពីការប្រើប្រាស់នូវ overload នៃប្រមាណវិធី assignment operator នៅក្នុង class iArray សំរាប់ assign object មួយនៃ class iArray ទៅអោយ object មួយទៀតនៃ class តែមួយ ។

```
// Header File: iarray.h
// Class iArray, representing array with dynamic size

#ifndef IARRAY_H
#define IARRAY_H

#include <iostream.h>

class iArray{
    private:
```

```
        int *elements;
        int size;

    public:
        iArray(int size=5);
        ~iArray();

        iArray& operator = (iArray& iArr);
        void copy(iArray& iArr);

        void setElement(int value, int index);
        int getElement(int index);

        void resetSize(int size);
        int getSize();
        void insert();
        void display();
        void sort();
};
iArray::iArray(int size) : size(size){
    if(this->size<=0) this->size = 5;
    elements = new int[size];
    for(int i=0; i<size; i++)
        elements[i] = 0;
}

iArray::~iArray(){ delete[] elements; }

void iArray::setElement(int value, int index){
    if(index<0 || index>=size){
        cout << "Index Out Of Bound." << endl;
    } else {
        elements[index] = value;
    }
}
int iArray::getElement(int index){
    if(index<0 || index>=0){
        cout << "Index Out Of Bound." << endl;
        return '\0';
    } else {
        return elements[index];
    }
}

iArray& iArray::operator = (iArray& iArr){
    this->copy(iArr);
    return *this;         // Return a reference to myself.
}
void iArray::copy(iArray& iArr){
    delete[] elements;
    size = iArr.size;
    elements = new int[size];
    for(int i=0; i<size; i++)
        elements[i] = iArr.elements[i];
}
```

```
void iArray::resetSize(int size){
     this->size = size;
     if(this->size<=0) this->size = 5;
     delete[] elements;
     elements = new int[size];
     for(int i=0; i<size; i++)
          elements[i] = 0;
}
int iArray::getSize(){ return size; }

void iArray::insert(){
     for(int i=0; i<size; i++){
          cout << "Elements[" << i << "]: ";
          cin >> elements[i];
     }
}
void iArray::display(){
     for(int i=0; i<size; i++){
          cout << elements[i];
     }
     cout << "\b\b";
}

void iArray::sort(){
     int temp;
     for(int i=0; i<size-1; i++)
          for(int j=i+1; j<size; j++)
               if(elements[i]>elements[j]){
               temp = elements[i];
               elements[i] = elements[j];
               elements[j] = temp;
               }
}

#endif

/* ------------------------------------------------------------------ */

// File: prog4_2.cpp
// Class iArray, representing array with dynamic size

#include <iarray.h>

void main(){

     iArray obj1(5);
     iArray obj2(3);
     iArray obj3(4);

     int values[5] = {7, 4, 5, 15, 3};

     for(int i=0; i<5; i++){
          obj1.setElement(values[i], i);
          if(k<3) obj2.setElement(values[i], i);
```

```
        if(k<4) obj3.setElement(values[i], i);
    }

    cout << "\nobj1: "; obj1.display();
    cout << "\nobj2: "; obj2.display();
    cout << "\nobj3: "; obj3.display();

    obj3 = obj2 = obj1;

    cout << "\n\nAssigning obj3 = obj2 = obj1";
    cout << "\nobj1: "; obj1.display();
    cout << "\nobj2: "; obj2.display();
    cout << "\nobj3: "; obj3.display();

    cout << "\n\nInput values for obj1\n";
    obj1.insert();

    cout << "\n\nAfter inputting values for obj1, "
         << "but obj2 and obj3 are still the same as the previous values";
    cout << "\nobj1: "; obj1.display();
    cout << "\nobj2: "; obj2.display();
    cout << "\nobj3: "; obj3.display();
}
```

4. Arithmetic Operators Overloading

Arithmetic operator គឺជាប្រភេទប្រមាណវិធីដែលត្រូវការអង្គប្រមាណវិធីពីរសំរាប់ការគណនា។ នៅក្នុង arithmetic operator រួមមាន operators ដូចជា: `*`, `/`, `%`, `+`, `-`។

ទំរង់ទូទៅនៃការប្រកាស overloading arithmetic operator គឺ:

```
ReturnType operator ArithmeticOperatorSign (parameters);
```

កម្មវិធីទី៣ សម្រាប់បញ្ជាក់ពីការប្រើប្រាស់កន្សោម arithmetic operator អនុវត្តទៅលើការ បូក ដក គុណ និងចែក ប្រភាគពីរ។

```
// Header File: rational.h
// Class Rational, representing arithmetic operator overloading

#ifndef RATIONAL_H
#define RATIONAL_H

#include <iostream.h>

class Rational{
    protected:
        double num;
        double den;

    public:
        Rational(double n = 1, double d = 1);
        Rational(Rational& arg);
```

```
        void setNumberator(double n);
        double getNumberator();

        void setDenominator(double d);
        double getDenominator();

        void insert();
        void display();

        Rational multiply(Rational& arg);
        Rational operator * (Rational& arg);

        Rational divide(Rational& arg);
        Rational operator / (Rational& arg);

        Rational add(Rational& arg);
        Rational operator + (Rational& arg);

        Rational subtract(Rational& arg);
        Rational operator - (Rational& arg);
};

Rational::Rational(double n, double d) : num(n), den(d){
    if(d==0){ den = 1; }
}

Rational::Rational(Rational& arg) : num(arg.num), den(arg.den){}

void Rational::setNumberator(double n){ num = n; }
double Rational::getNumberator(){ return num; }

void Rational::setDenominator(double d){
    if(d==0){ den = 1; }
}
double Rational::getDenominator(){ return den; }

void Rational::insert(){
    cout << endl << "Enter Numberator: "; cin >> num;
    cout << endl << "Enter Denominator: "; cin >> den;
    while(den==0){
        cin >> den;
        if(den==0){
            cout << endl << "Re-enter (Denominator can not be zero): ";
        }
    }
}
void Rational::display(){
    cout << "(" << num << "/" << den << ")" << endl;
}

Rational Rational::multiply(Rational& arg){
    Rational temp;
    temp.num = num * arg.num;
    temp.den = den * arg.den;
    return temp;
```

```
}
Rational Rational::operator * (Rational& arg){
     Rational temp;
     temp.num = num * arg.num;
     temp.den = den * arg.den;
     return temp;
}

Rational Rational::divide(Rational& arg){
     Rational temp;
     temp.num = num * arg.den;
     temp.den = den * arg.num;
     return temp;
}
Rational Rational::operator / (Rational& arg){
     Rational temp;
     temp.num = num * arg.den;
     temp.den = den * arg.num;
     return temp;
}

Rational Rational::add(Rational& arg){
     Rational temp;
     temp.num = num * arg.den + den * arg.num;
     temp.den = den * arg.den;
     return temp;
}

Rational Rational::operator + (Rational& arg){
     Rational temp;
     temp.num = num * arg.den + den * arg.num;
     temp.den = den * arg.den;
     return temp;
}

Rational Rational::subtract(Rational& arg){
     Rational temp;
     temp.num = num * arg.den - den * arg.num;
     temp.den = den * arg.den;
     return temp;
}
Rational Rational::operator - (Rational& arg){
     Rational temp;
     temp.num = num * arg.den - den * arg.num;
     temp.den = den * arg.den;
     return temp;
}

#endif

/* ------------------------------------------------------------ */

/*
File: prog4_3.cpp
```

```
Class Rational, representing multiplication, division, addition and
subtraction between two fractions
*/

#include <rational.h>

void main() {

    Rational obj1;
    Rational obj2(5, 2);
    Rational obj3(10, 3);

    cout << "obj1 is "; obj1.display();
    cout << "obj2 is "; obj2.display();
    cout << "obj3 is "; obj3.display();

    cout << endl;
    obj1 = obj2 * obj3;
    cout << "obj2 * obj3 = "; obj1.display();

    obj1 = obj2 / obj3;
    cout << "obj2 / obj3 = "; obj1.display();

    obj1 = obj2 + obj3;
    cout << "obj2 + obj3 = "; obj1.display();

    obj1 = obj2 - obj3;
    cout << "obj2 - obj3 = "; obj1.display();
}
```

មេរៀនទី ១០: POLYMORPHISM

Object-Oriented Programming មានសមត្ថភាពក្នុងការអនុវត្តបានច្រើនទំរង់។ សមត្ថភាពនេះត្រូវបានគេហៅថា Polymorphism។ Polymorphism វាកើតឡើងនៅពេលដែល member functions នៅក្នុង class hierarchy មួយប្រព្រឹត្តផ្សេងៗគ្នា ឬអាស្រយ័ទៅលើ object ដែលអនុវត្ត message។

1. What is Polymorphism?

Polymorphism គឺកើតឡើងពីពាក្យក្រិច poly ដែលមាននយ័ថាច្រើន ផ្សំជាមួយ និងពាក្យក្រិច morphism ដែលមាននយ័ថាទំរង់។ ហេតុនេះ មាននយ័ថាច្រើនទំរង់។ នៅក្នុង OOP, Polymorphism គេសំដៅទៅរក member functions ទាំងឡាយមានឈ្មោះដូចគ្នានៅក្នុង class hierarchy មានការប្រព្រឹត្តផ្សេងៗគ្នាអាស្រយ័ទៅតាម object ដែលវានីមួយៗសំដៅទៅរក។ ហេតុនេះ polymorphism គឺជាដំណើរនៃការកំនត់ចេញ objects មួយចំនួននៃ class ផ្សេងគ្នានៅក្នុង class hierarchy អោយទៅជាក្រុមមួយ ហើយហៅ methods ទាំងឡាយអោយអនុវត្ត operations ផ្សេងៗគ្នារបស់ object ទាំងនេះរៀងគ្នាដោយប្រើប្រាស់ message passing រួម។ ជាឧទាហរណ៍សន្មតថាគេមាន class **Cube** មួយទទួលមរតកពី class **Rectangle** ហើយ classes ទាំងពីរសុទ្ធតែមាន method **draw()** ផ្ទាល់ខ្លួនរៀងគ្នាសំរាប់គូរចេញជារាងរបស់វានីមួយៗ។ នៅពេលដែលគេត្រូវគូរ ចេញជារាងណាមួយក្នុងចំនោមពួកវា គេត្រូវកំនត់ថាតើគេកំពុងតែនិយាយសំដៅទៅរក object របស់ class ណាមួយក្នុងចំនោម classes ទាំងពីរ ព្រោះថាគេបានប្រើ message passing ជា **draw()** តែមួយ។

2. Early Binding

Early Binding (គេអាចហៅបានថាជា static binding or static linkage) គឺជាការកំនត់ជ្រើសរើសទុកនៅក្នុងពេល compile នូវឈ្មោះរបស់ method សំរាប់យកមកអនុវត្តក្នុងពេលដែល code ត្រូវបាន executed។ នៅក្នុងពេលដែល compile, C++ compiler បានកំនត់ជ្រើសរើសទុក function ណាមួយ (ក្នុងចំនោម functions ទាំងឡាយដែលមានឈ្មោះដូចគ្នា) សំរាប់ការអនុវត្តអាស្រ័យទៅតាម arguments ដែលបញ្ជូនទៅអោយ parameters របស់ functions។

ឧទាហរណ៍ទី១: ការកំនត់ជ្រើសទុកក្នុងពេល compile នូវ functions មកអនុវត្ត

```
class Shape{
    protected:
        double width;
        double height;

    public:
        Shape(double width = 0, double height = 0)
```

```
            : width(width), height(height){}

        void setLength(double width = 0, double height = 0){
            this.width = width;
            this.height = height;
        }
        double area(){ return 0; }
        void display(){
            cout << "Parent class area: " << area() << endl;
        }
};

class Rectangle : public Shape{
    public:
        Rectangle(double width = 0, double height = 0)
            : Shape(width, height){}

        double area (){ return (width * height); }
        void display(){
            cout << "Display Information about Rectangle: " << endl;
            cout << "Width : " << width << endl;
            cout << "Height: " << height << endl;
            cout << "Rectangle class area: " << area() << endl;
        }
};
```

តាមឧទាហរណ៍ទី១នេះគេបាន class Shape គឺជា base class របស់ class Rectangle។ class Shape មាន member functions:

- double area(): សំរាប់ return តំលៃ 0 អោយវា ដោយសារវាមិនបានបញ្ជាក់ពីរាងណាមួយទេ
- void display(): គឺសំរាប់បង្ហាញនូវពត៌មានរបស់ក្រឡាផ្ទៃរបស់ shape
- void setLength():គឺសំរាប់កំណត់នូវតំលៃរបស់ data members ទាំងពីរគឺ width និង height

ហើយ class Rectangle ដែលជា Derived class ដែលបានទទួលមរតកមកពី class Shape មាន function members:

- function area(), display() និង setLength() គឺជា function ដែលទទួលបានមកពី Base class
- double area(): គឺជា function ផ្ទាល់របស់វា សំរាប់គណនានូវក្រឡាផ្ទៃរបស់ចតុកោណកែង
- void display(): គឺជា function ផ្ទាល់របស់វា សំរាប់បង្ហាញនូវពត៌មានរបស់ចតុកោណកែង

នៅក្នុងពេលដែល compile ទៅលើ class Shape, compiler បានជ្រើសទុកជាមុននូវ functions: area(), display(), setLength() សំរាប់អនុវត្តជាមួយនឹង object, reference to object និង pointer to object របស់ Base class។ ចំនែកឯ class Rectangle ពេលត្រូវបាន compile, compiler បានជ្រើសទុក functions setLength() ទទួលបានមកពី class Shape, រីឯ functions: area() និង display() ជា function ផ្ទាល់ខ្លួន សំរាប់អនុវត្តជាមួយ នឹង object, reference to object និង pointer to object របស់ខ្លួនវា។ ដូចនេះបើសិនជាគេមាន object, pointer to object ដូចខាងក្រោម:

```
    Shape *shape;
    Rectangle rect(10, 7);
    shape = &rect;
```

នោះ statements ខាងក្រោម:

- shape.setLength(10.0, 15.0): គឺហៅ function setLength() របស់ calss Shape មកអនុវត្ត ដោយផ្តល់តំលៃ 10.0 ទៅអោយ width និង 15.0 ទៅអោយ height
- shape.area(): គឺហៅ function area() ផ្ទាល់របស់ calss Shape មកអនុវត្ត
- shape.display(): គឺហៅ function display () ផ្ទាល់របស់ calss Shape មកអនុវត្ត
- rect.setLength(5.0, 7.5): គឺហៅ function setLength() របស់ calss Rectangle ដែលទទួលបានមកពី calss Shape មកអនុវត្ត ដោយផ្តល់តំលៃ 5.0 ទៅអោយ width និង 7.5 ទៅអោយ height
- rect.area(): គឺហៅ function area() ផ្ទាល់របស់ calss Rectangle មកអនុវត្ត
- rect.display(): គឺហៅ function display() ផ្ទាល់របស់ calss Rectangle មកអនុវត្ត
- shape->setLength(8.0, 4.5): គឺហៅ function setLength() របស់ calss Rectangle ដែលទទួលបានមកពី calss Shape មកអនុវត្ត ដោយផ្តល់តំលៃ 8.0 ទៅអោយ width និង 4.5 ទៅអោយ height
- shape->area(): គឺហៅ function area() របស់ calss Rectangle ដែលទទួលបានមកពី calss Shape មកអនុវត្ត
- shape->display(): គឺហៅ function display() របស់ calss Rectangle ដែលទទួលបានមកពី calss Shape មកអនុវត្ត

កម្មវិធីទី១: កម្មវិធីនេះបង្ហាញពីការប្រើប្រាស់ functions តាមទំរង់ Early Binding

```
#include <iostream.h>

class Shape {
    protected:
        double width;
        double height;

    public:
        Shape(double width = 0, double height = 0)
            : width(width), height(height){}

        double area(){ return 0; }
        double display(){
            cout << "Parent class area: " << area() << endl;
        }
};

class Rectangle : public Shape{
```

```
    public:
        Rectangle(double width = 0, double height = 0)
            : Shape(width, height){}

        double area (){ return (width * height); }
        void display(){
            cout << "Display Information about Rectangle: " << endl;
            cout << "Width : " << width << endl;
            cout << "Height: " << height << endl;
            cout << "Rectangle class area: " << area() << endl;
        }
};

class Triangle : public Shape{
    public:
        Triangle(double width = 0, double height = 0)
            : Shape(width, height){}

        double area (){ return (width * height / 2); }
        void display(){
            cout << "Display Information about Triangle: " << endl;
            cout << "Width : " << width << endl;
            cout << "Height: " << height << endl;
            cout << "Triangle class area: " << area() << endl;
        }
};

void main( ){
    Shape *shape;
    Rectangle rec(10, 7);
    Triangle tri(10, 5);

    rec.display();        // call method display in class Rectangle
    cout << endl;

    tri.display();        // call method display in class Triangle
    cout << endl;

    shape = &rec;         // store the address of Rectangle
    shape->display();     // call method display in class Shape
}
```

3. Virtual Functions

Virtual functions គឺជា member functions ដែលមិនលេចរូបរាង ដើម្បីអោយមានការជ្រើសរើសទុកក្នុងពេល compile សំរាប់យកមកអនុវត្តនោះទេ តែវាកើតមានរូបនៅក្នុងពេល run នៅត្រង់ផ្នែកខ្លះនៃ code ពេលមានការអនុវត្តហៅ member functions ទាំងនេះមកអនុវត្ត។ វាផ្តល់ប្រយោជន៍ដល់ការជ្រើសរើសយក function មកអនុវត្តក្នុងពេល run ទៅតាម objects ដែលគេសំដៅទៅរក។ ដើម្បីអោយ function member មួយក្លាយជា virtual function គេគ្រាន់តែបន្ថែម keyword

virtual នាំ protype របស់វាជាស្រេច។ Non-member functions, static member functions, និង constructors មិនអាចជា virtual functions បានទេ។

ឧទាហរណ៍ទី២: បញ្ជាក់ពីការប្រកាស virtual functions

```
class Account{
    private:
        double balance;

    public:
        Account(double balance = 100) : balance(balance){}
        virtual double getBalance(){ return balance; }
        virtual void displayBalance(){
            cout << "Balance not available for base type." << endl;
        }
};
```

នៅក្នុងឧទាហរណ៍ទី២នេះ functions getBalance() និង displayBalance() គឺជា virtual functions។ សំរាប់ការ definitions របស់ virtual functions គេអនុវត្តដូចជា member functions ដែរ ហើយក៏ពុំមាន keyword virtual នាំមុខទៀតទេ។

4. Late Binding

Late Binding (គេអាចហៅបានថាជា dynamic binding or dynamic linkage) គឺជាការកំនត់ជ្រើសរើស functions មកអនុវត្តនៅក្នុងពេល run។ នៅក្នុងករណីដែលមិនមានការជ្រើសរើសទុកនូវ functions សំរាប់យកមកអនុវត្តនៅក្នុងពេល compile នោះទេ ការសំរេចយក functions មកអនុវត្តត្រូវកំនត់នៅក្នុងពេល run ដោយធ្វើការសំរេចយក functions របស់ object ដែលគេកំពុងសំដៅទៅរក។ Late Binding ត្រូវអនុវត្តតាមរយៈ virtual functions ព្រោះថាពួកវាមិនមានរូបរាងនៅក្នុងពេល compile សំរាប់ធ្វើការកំនត់ជ្រើសទុក functions មកអនុវត្តដោយ compiler តាមលក្ខណៈ Early Binding នោះទេ។

ឧទាហរណ៍ទី៣: ការកំនត់ជ្រើសទុកក្នុងពេល compile នូវ functions មកអនុវត្ត

```
class Shape{
    protected:
        double width;
        double height;

    public:
        Shape(double width = 0, double height = 0)
            : width(width), height(height){}

        void setLength(double width = 0, double height = 0){
            this.width = width;
            this.height = height;
```

```
        }

        virtual double area(){ return 0; }
        virtual void display(){
            cout << "Parent class area: " << area() << endl;
        }
};

class Rectangle : public Shape{
    public:
        Rectangle(double width = 0, double height = 0)
            : Shape(width, height){}

        virtual double area (){ return (width * height); }
        virtual void display(){
            cout << "Display Information about Rectangle: " << endl;
            cout << "Width : " << width << endl;
            cout << "Height: " << height << endl;
            cout << "Rectangle class area: " << area() << endl;
        }
};
```

តាមឧទាហរណ៍ទី៣នេះគេបាន class Shape គឺជា base class របស់ class Rectangle។ class Shape មាន member functions:

- double area(): សំរាប់ return តំលៃ 0 អោយវា ដោយសារវាមិនបានបញ្ជាក់ពីរាងណាមួយទេ
- void display(): គឺសំរាប់បង្ហាញនូវពត៌មានរបស់ក្រឡាផ្ទៃរបស់ shape
- void setLength(): សំរាប់កំណត់នូវតំលៃរបស់ data members ទាំងពីរគឺ width និង height

ហើយ class Rectangle ដែលជា Derived class ដែលបានទទួលមរតកមកពី class Shape មាន function members:

- function area(), display() និង setLength() គឺជា function ដែលទទួលបានមកពី Base class
- double area(): គឺជា function ផ្ទាល់របស់វា សំរាប់គណនានូវក្រឡាផ្ទៃរបស់ចតុកោណកែង
- void display(): គឺជា function ផ្ទាល់របស់វា សំរាប់បង្ហាញនូវពត៌មានរបស់ចតុកោណកែង

នៅក្នុងពេលដែល compile ទៅលើ class Shape, compiler បានជ្រើសទុកជាមុននូវ functions: setLength() សំរាប់អនុវត្តជាមួយនឹង object, reference to object និង pointer to object របស់ Base class។ ចំនែកឯ class Rectangle ពេលត្រូវបាន compile, compiler បានជ្រើសទុក functions setLength() ទទួលបានមកពី class Shape តែប៉ុណ្ណោះសំរាប់យកមកអនុវត្តជាមួយនឹង object, reference to object និង pointer to object របស់ខ្លួនវា, រីឯ functions: area() និង display() ជា function របស់វាផ្តល់លទ្ធភាព សំរាប់ការជ្រើសរើសយកមកអនុវត្តនៅក្នុងពេល run។ ដូចនេះបើសិនជាគេមាន object, pointer to object ដូចខាងក្រោម:

```
    Shape *shape();
```

```
    Rectangle rect(10, 7);
    shape = &rect;
```

នោះ statements ខាងក្រោម:

- shape.setLength(10.0, 15.0): គឺហៅ function setLength()របស់ calss Shape មកអនុវត្ត ដោយផ្តល់តំលៃ 10.0 ទៅអោយ width និង 15.0 ទៅអោយ height
- shape.area(): គឺហៅ function area() ផ្ទាល់របស់ calss Shape មកអនុវត្ត
- shape.display(): គឺហៅ function display () ផ្ទាល់របស់ calss Shape មកអនុវត្ត
- rect.setLength(5.0, 7.5): គឺហៅ function setLength() របស់ calss Rectangle ដែលទទួលបានមកពី calss Shape មកអនុវត្ត ដោយផ្តល់តំលៃ 5.0 ទៅអោយ width និង 7.5 ទៅអោយ height
- rect.area(): គឺហៅ function area() ផ្ទាល់របស់ calss Rectangle មកអនុវត្ត
- rect.display(): គឺហៅ function display() ផ្ទាល់របស់ calss Rectangle មកអនុវត្ត
- shape->setLength(8.0, 4.5): គឺហៅ function setLength() របស់ calss Rectangle ដែលទទួលបានមកពី calss Shape មកអនុវត្ត ដោយផ្តល់តំលៃ 8.0 ទៅអោយ width និង 4.5 ទៅអោយ height
- shape->area(): គឺហៅ function area() ផ្ទាល់របស់ calss Rectangle មកអនុវត្ត
- shape->display(): គឺហៅ function display() ផ្ទាល់របស់ calss Rectangle មកអនុវត្ត

កម្មវិធីទី២: កម្មវិធីនេះបង្ហាញពីការប្រើប្រាស់ virtual functions តាម Late Binding

```
#include <iostream.h>

class Account{
    private:
        double balance;

    public:
        Account(double balance) : balance(balance){}
        virtual double getBalance(){ return balance; }
        virtual void displayBalance(){
            cout << "Balance not available for base type." << endl;
        }
};

class CheckingAccount : public Account{
    public:
        CheckingAccount(double balance) : Account(balance){}
        void displayBalance(){
            cout << "Checking account balance: " << getBalance() << endl;
        }
};

class SavingsAccount : public Account{
```

```
    public:
        SavingsAccount(double balance) : Account(balance){}
        void displayBalance(){
            cout << "Savings account balance: " << getBalance() << endl;
        }
};

void main(){

    // Create objects of type CheckingAccount and SavingsAccount.
    CheckingAccount *pChecking = new CheckingAccount(500.0);
    SavingsAccount *pSavings = new SavingsAccount(1000.0);

    // Create object of type Account
    Account *pAccount;

    // Call displayBalance using a pointer to Account.
    pAccount = pChecking;
    pAccount->displayBalance();

    // Call display Balance using a pointer to Account.
    pAccount = pSavings;
    pAccount->displayBalance();
}
```

មេរៀនទី ១១: Templates

Chapter នេះយើងសិក្សាអំពីលក្ខណៈថ្មីរបស់ Object-Oriented Programming ដែលមានលក្ខណៈ flexible សម្រាប់ដោះស្រាយបញ្ហា។ ចំនុចដែលសំខាន់ៗនៅក្នុងមេរៀននេះរួមមាន:

- ការបង្កើតចេញនូវ function មួយសម្រាប់តាងឲ្យ functions ទាំងឡាយដែលមានលក្ខណៈការងារស្រដៀងគ្នា function នោះត្រូវបានគេហៅថា function templatele
- ការបង្កើតចេញនូវ class មួយសម្រាប់តាងឲ្យ classes ទាំងឡាយដែលមានលក្ខណៈទិន្នន័យដូចគ្នា class នោះត្រូវបានគេហៅថា class templatele

1. Function Templates:

Function template គឺជាវិធីសាស្ត្រមួយសម្រាប់កំណត់ចេញនូវ function មួយសម្រាស់ជំនួសឲ្យគ្រួសារនៃ functions ទាំងឡាយ ដែលស្ថិតនៅក្នុងលក្ខណៈការងារស្រដៀងគ្នា។ នៅក្នុង function template នេះយ៉ាងហោចណាស់មាន formal parameter មួយដែលមានប្រភេទទិន្នន័យទូទៅ (generic type)។

នៅក្នុងមេរៀនទី៤ យើងបានសិក្សាអំពីប្រភេទនៃ function overloading រួចហើយ ដែល functions ទាំងនោះមានប្រភេទ operations ស្រដៀងៗគ្នា និងអាចហៅ functions ទាំងនោះមកអនុវត្តនៅក្នុងកម្មវិធីណាមួយ។ នៅក្នុង function overloading ទោះបីជាឈ្មោះរបស់ functions ទាំងនោះមានឈ្មោះដូចគ្នាក៏ដោយ ក៏ codes ទាំងឡាយត្រូវបានគេសរសេរឡើងវិញម្តងហើយម្តងទៀតសម្រាប់ function នីមួយៗ។

នៅក្នុងភាសា C++ ផ្តល់ឲ្យយើងនូវលក្ខណៈពិសេសដែលមានសមត្ថភាពក្នុងការកំណត់ function តែមួយសម្រាប់តាងឲ្យគ្រួសារនៃ functions មានលក្ខណៈស្រដៀងគ្នា។ នៅពេល function មួយត្រូវបានគេកំណត់ឡើងសម្រាប់តាងឲ្យគ្រួសារនៃ functions វាត្រូវបានគេហៅថា function template។ ផលប្រយោជន៍ដែលបានមកពីការប្រើប្រាស់ function templates គឺជៀសវាងកើតមានការច្រំដែលចំពោះការសរសេរ code។ Function template មិនបានកំណត់ប្រភេទទិន្នន័យពិតរបស់ arguments ដែលវាត្រូវទទួលយកនោះទេ ព្រោះថា formal parameters របស់វាមានប្រភេទទិន្នន័យទូទៅ (generic or parameterized type)។ ពេលហៅយកមកអនុវត្ត ប្រភេទទិន្នន័យទូទៅត្រូវកំណត់ឡើងប្រភេទទិន្នន័យរបស់ arguments ដែលវាទទួលយក។

ទំរង់ទូទៅនៃការប្រកាស function template គឺ:

```
template <class PType1 [, class PType2, ...]>
```

```
ReturnType functionName(parameters);
```

- template និង class គឺជា keyword
- PType1, PType2, ... គឺជាប្រភេទទិន្នន័យទូទៅរបស់ parameter មួយរបស់ function template
- ReturnType គឺជា return type របស់ function template។ វាក៏អាចជា PType1, PType2, ... ផងដែរ
- functionName គឺជាឈ្មោះរបស់ function template
- parameters គឺជា parameters របស់ function template ដែលក្នុងនោះយ៉ាងហោចណាស់មាន parameter មួយមានប្រភេទទិន្នន័យជា PType1, PType2, ...

ឧទាហរណ៍ទី១: Function Template សម្រាប់តាងអោយរាល់ប្រភេទអនគមន៍ ដើម្បីធ្វើការផ្លាស់ប្តូរតម្លៃរវាងអញ្ញាតិពីរទៅវិញទៅមកប្រើប្រាស់ constructor របស់ class **ofstream**

```
template <class Type>
void swap(Type& first, Type& second){
    Type temp;

    temp = first;
    first = second;
    second = temp;
}
```

តាមរយៈឧទាហរណ៍ទី១ នេះគេបានប្រកាសចេញ function template មានឈ្មោះថា swap វាត្រូវការប្រភេទទិន្នន័យជា Type ហើយវាមាន return type ជា void។ Statements នៅក្នុង body របស់ function template នេះអនុវត្តធ្វើការផ្លាស់ប្តូរតម្លៃទៅវិញទៅមករវាងអញ្ញាតិទូទៅពីរ។ តាមរយៈ function template នេះគេអាចហៅវាយកមកអនុវត្តជាមួយនឹងគូអញ្ញាតិដូចខាងក្រោម:

```
int x=34, y=83;
float a=10.5, b=85.75;
```

`swap(x, y);` មាននយ័ថាវានឹងធ្វើការផ្លាស់ប្តូរតម្លៃរវាងអញ្ញាតិចំនួនគត់ x, y ទាំងពីរទៅវិញទៅមក

`swap(a, b);` មាននយ័ថាវានឹងធ្វើការផ្លាស់ប្តូរតម្លៃរវាងអញ្ញាតិចំនួនទសភាគ a, b ទាំងពីរទៅវិញទៅមក

កម្មវិធីទី១: កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ function template សម្រាប់ជំនួសឲ្យ functions ទាំងឡាយដែលមានការងារស្រដៀងគ្នា។

```
#include <iostream.h>
```

```
template <class T>
void swap(T& x, T& y) {
     T temp = x;
     x = y;
     y = temp;
}

void main() {
     int a, b;
     float c, d;

     cout << "Enter A, B values (integer): "; cin >> a >> b;

     cout << "A and B before swaping : " << a << "\t" << b << endl;
     swap(a, b);
     cout << "A and B after swaping : " << a << "\t" << b << endl;

     cout << "Enter C, D values (float): "; cin >> c >> d;

     cout << "C and D before swaping : " << c << "\t" << d << endl;
     swap(a, b);
     cout << "C and D after swaping : " << c << "\t" << d << endl;
}
```

កម្មវិធីទី២: កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ function template សម្រាប់ជំនួសឲ្យ functions ទាំងឡាយដែលមានការងារស្រដៀងគ្នា។

```
#include <iostream.h>

template <class T>
T Absolute(T number){
     return (number > 0) ? number : -number;
}

void main(){
     int a = 7, b = -7;
     cout << "Absolute value of " << a << " = " << Absolute(a) << endl;
     cout << "Absolute value of " << b << " = " << Absolute(b) << endl;

     double x = 7.0923, y = -7.0923;
     cout << "Absolute value of " << x << " = " << Absolute(x) << endl;
     cout << "Absolute value of " << y << " = " << Absolute(y) << endl;
}
```

កម្មវិធីទី៣: កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ function template សម្រាប់ដើម្បីធ្វើការតម្រៀបទិន្នន័យទៅតាមលំដាប់កើន ដោយប្រើប្រាស់វិធីសាស្ត្រ quick sort។

```
#include <iostream.h>
#include <stdlib.h>

#define size 10

template <class T>
```

```
void quickSort(T a[], const int& leftarg, const int& rightarg) {
    if (leftarg < rightarg) {
        T pivotvalue = a[leftarg];
        int left = leftarg - 1;
        int right = rightarg + 1;

        for( ; ; ) {
            while (a[--right] > pivotvalue);
            while (a[++left] < pivotvalue);

            if (left >= right) break;

            T temp = a[right];
            a[right] = a[left];
            a[left] = temp;
        }

        int pivot = right;
        quickSort(a, leftarg, pivot);
        quickSort(a, pivot + 1, rightarg);
    }
}

int main(void) {
    int sortme[size], i;

    cout << "Before call function template quickSort: " << endl;
    randomize();
    for (i = 0; i < size; i++) {
        sortme[i] = random(100);
        cout << sortme[i] << ", ";
    }
    cout << "\b\b" << endl;

    quickSort(sortme, 0, size - 1);
    cout << "\nAfter call function template quickSort: " << endl;
    for (i = 0; i < size; i++) cout << sortme[i] << ", ";
    cout << "\b\b" << endl;
    return 0;
}
```

2. Class Templates:

បន្ថែមទៅលើ function templates, C++ ក៏បានផ្តល់ឲ្យយើងនូវ concept របស់ class templates។ ជានិយមនយ័ class template គឺជាការកំណត់ class មួយដែលអធិប្បាយពីគ្រួសាររបស់ classes ដែលទាក់ទងគ្នា។ ភាសា C++ ផ្តល់ឲ្យ users នូវសមត្ថភាពបង្កើតចេញ class មួយមានផ្ទុកទិន្នន័យប្រភេទជា generic ឬ parameterized types។ បែបបទនៃការប្រកាស class template វាមានលក្ខណៈដូចគ្នាទៅនឹង function templates ដែរ។

ទំរង់ទូទៅនៃការប្រកាស class template គឺ:

```
template <class PType1 [, PType2, ...]>
```

```
class TemplateClassName[:derivation ClassBaseList]{
    ......
    PType1 data1 [, data12, ...];
    [PType2 data2 [, data22, ...]];
    ......
    [function member;]
    ......
};
```

- template និង class គឺជា keyword
- PType1, PType2, ... គឺជាប្រភេទទិន្នន័យទូទៅរបស់ parameter មួយរបស់ function template
- TemplateClassNaem គឺជាឈ្មោះរបស់ class template ដែលអាចទទួលមរតកមកពី base class (super class)
- Derivation វាអាចជា private, protected ឬ public វាបញ្ជាក់ពីមធ្យោបាយក្នុងការទទួលយកមរតកមកពី base class ផ្សេងៗ
- ClassBaseList គឺជា base classes ទាំងឡាយដែលត្រូវផ្តល់មរតកទៅឲ្យ class template ហើយពួកវាត្រូវផ្តាច់ចេញពីគ្នាដោយសញ្ញា comma
- PType1 data1 [, data12, ...]; គឺបញ្ជាក់ថា class template មានទិន្នន័យ data1 និង data12 មានប្រភេទទិន្នន័យជា generic type ជា PType1។
- [PType2 data2 [, data22, ...]]; គឺបញ្ជាក់ថា class template មានទិន្នន័យ data2 និង data22 មានប្រភេទទិន្នន័យជា generic type ជា PType2។

នៅពេលដែល class template មួយបានកំណត់ឡើងរួចរាល់ហើយ វាទាមទារឲ្យយើងបង្កើតចេញនូវ objects របស់វាដោយប្រើប្រាស់ប្រភេទទិន្នន័យ primitive type ឬ user-defined type សម្រាប់ជំនួសប្រភេទទិន្នន័យទូទៅដែលប្រើប្រាស់នៅក្នុង class template។

ទំរង់ទូទៅរបស់ class template សប្រាប់អនុវត្តធ្វើជាប្រភេទទិន្នន័យរបស់ object កំណត់ដោយ:

```
TemplateClassName <ExactType1 [, ExacType2, ...]>
```

ដោយក្នុងនោះ ExactType1 , ExacType2, ... គឺជា primitive types ឬ user-defined types ផ្សេងៗដែលត្រូវជំនួសអោយប្រភេទទិន្នន័យទូទៅរបស់ data ខ្លះៗនៅក្នុង class template។

ឧទាហរណ៍ទី២: Function Template សម្រាប់តាងអោយរាល់ប្រភេទអនគមន៍ ដើម្បីធ្វើការផ្លាស់ប្តូរ

តម្លៃរវាងអញ្ញាតិពីរទៅវិញទៅមកប្រើប្រាស់ constructor របស់ class **ofstream**

```
template <class Element>
class Array{
    public:
        Array() : arrayLen(0), numElements(0), array(0) {}

        void add(Element e){
            if(numElements == arrayLen) makeRoom();
            array[numElements++] = e;
        }

        Element operator[](int n) const{
            checkSubscript(n);
            return array[n];
        }

        int length() const { return(numElements); }

    private:
        Element* array;  // Pointer to an array of objects
        int arrayLen;    // Length of the array currently allocated
        int numElements; // The number of elements in the array
        void checkSubscript(int n) const;
        void makeRoom();
        static int chunkSize;
};
```

នៅពេលដែលគេធ្វើការកំនត់ definition របស់ member functions មួយនៅខាងក្រៅ class template គេត្រូវប្រើប្រាស់ template keyword និង formal parameter list របស់ class template។

ការកំនត់ definition របស់ member functions មួយនៅក្រៅ class មានទំរង់ដូចខាងក្រោម:

```
template <class Element>
void Array<Element>::makeRoom(){}
```

ខាងក្រោមគឺជាទំរង់នៃការកំណត់តម្លៃចាប់ផ្តើមទៅឲ្យ static data member

```
template <class E> Array<E>::chunkSize = 10;
```

តាមរយៈឧទាហរណ៍ទី២ នេះគេបានប្រកាសចេញ class template តាងឲ្យគ្រួសារនៃ classes ទាំងឡាយដែលមានប្រភេទទិន្នន័យទូទៅមួយ ហើយ objects របស់ classes ទាំងនោះតែមាន methods: add(), operator() និង length()។ Objects របស់ class template ដែលបានបង្កើតឡើងតាមការប្រកាសផ្សេងៗខាងលើ មាននយ័ដូចខាងក្រោម:

```
int x=34, y=83;
float a=10.5, b=85.75;
```

`swap(x, y);` មាននយ័ថាវានឹងធ្វើការផ្លាស់ប្តូរតម្លៃរវាងអញ្ញាតិចំនួនគត់ x, y ទាំងពីរទៅវិញទៅមក

`swap(a, b);` មាននយ័ថាវានឹងធ្វើការផ្លាស់ប្តូរតម្លៃរវាងអញ្ញាតិចំនួនទសភាគ a, b ទាំងពីរទៅវិញទៅមក

កម្មវិធីទី៤: កម្មវិធីបញ្ជាក់ពីការប្រើប្រាស់ class templat សម្រាប់ជំនួសឲ្យ classes ទាំងឡាយដែលមានការងារស្រដៀងគ្នា និងការអនុវត្តលើ objects របស់ class template។

```
// Header File: ArrayList.h

#include <iostream.h>

#ifndef ARRAYLIST_H
#define ARRAYLIST_H

template <class Type>
class ArrayList {
    private:
        Type* array;
        int capacity;
        int size;

    public:
        ~ArrayList();
        ArrayList(int capacity = 10);
        void display() const;
        int insert(const Type& element);
        int retrieve(Type& element, int index) const;
        int remove(Type& element, int index);
        Type getLargest() const;
        Type getSmallest() const;
        int getCapacity() const;
        int getSize() const;
};

template <class Type>
ArrayList <Type> :: ~ArrayList(){
    delete [] array;
    array = NULL;
    capacity = 0;
    size = 0;
}

template <class Type>
ArrayList <Type> :: ArrayList(int capacity){
    array = new Type[capacity];
    this->capacity = capacity;
    size = 0;
}

template <class Type>
```

```
void ArrayList <Type> :: display() const{
    cout << "Current values in list: " << endl;
    for(int count = 0; count < size; count++){
        cout << array[count] << ", ";
    }
    cout << "\b\b\n";
}

template <class Type>
int ArrayList <Type> :: insert(const Type& element){
    int success = 0;

    if(size == capacity){
        int newSize = capacity + (capacity / 2);
        Type* tempArray = new Type[newSize];

        for (int count = 0; count < size; count++ )
            tempArray[count] = array[count];

        delete [] array;
        array = tempArray;
        capacity = newSize;
    }

    array[size] = element;
    size++;

    success = 1;
    return success;
}

template <class Type>
int ArrayList <Type> :: retrieve(Type& element, int index) const{
    int success = 0;
    element = array[index];

    success = 1;
    return success;
}

template <class Type>
int ArrayList <Type> :: remove(Type& element, int index){
    int success = 0;

    if(index >= 0 && index < size){
        element = array[index];

        for(int count = index; count < size; count++){
            array[count] = *(array + (count + 1));
        }
        size--;
    }

    success = 1;
    return success;
```

```
}

template <class Type>
Type ArrayList <Type> :: getLargest() const{
    Type highest = array[0];

    for(int count = 1; count < size; count++){
        if (*(array + count) > highest)
            highest = *(array + count);
    }
    return highest;
}

template <class Type>
Type ArrayList <Type> :: getSmallest() const{
    Type lowest = array[0];

    for(int count = 1; count < size; count++){
        if (*(array + count) < lowest)
            lowest = *(array + count);
    }
    return lowest;
}

template <class Type>
int ArrayList <Type> :: getCapacity() const{
    return capacity;
}

template <class Type>
int ArrayList <Type> :: getSize() const{
    return size;
}

#endif

/* ------------------------------------------------------------- */

// File: ArrayList.cpp

#include <ArrayList.h>
#include <iostream.h>
#include <string.h>
#include <arraylis.h>

int main(){
    ArrayList <double> myList;

    int index;
    double value;

    myList.insert(1.7);
    myList.insert(2.3);
    myList.insert(-4.8);
    myList.insert(147.2);
```

```
    myList.insert(27.6);
    myList.insert(38.4);

    cout << "What is the index of this value? "; cin >> index;

    if(myList.remove(value, index))
        cout << "Good Job." << " Value removed is " << value
             << ", at index " << index << endl;
    else
        cout << "Invalid subscript." << endl;

    myList.display();

    if(myList.retrieve(value, index))
        cout << "\nRetrieval was successful." << endl;
    else
        cout << "Retrieval failed." << endl;

    double biggest = myList.getLargest();
    cout << "\nLargest value in list is: " << biggest << endl;

    double smallest = myList.getSmallest();
    cout << "\nSmallest value in list is: " << smallest << endl;

    int capacity = myList.getCapacity();
    cout << "\nThe capacity of the list is currently set to "
         << capacity << "." << endl;

    int size = myList.getSize();
    cout << "\nThere are currently " << size << " value(s) in the list."
         << endl;
}
```

មេរៀនទី ១២: Data File Operations

Chapter នេះសិក្សាអំពីការបង្កើត និងការ access ពី files ផ្ទុកនៅក្នុង Secondary Storage Devices។ ភាសា C++ ផ្តល់ឱ្យជាប្រភេទផ្សេងៗដែលអាច access ទៅលើ files ទៅតាមទំរង់ដើមរបស់ទិន្នន័យ។ ចំនុចសំខាន់ៗដែលត្រូវលើកយកមកសិក្សានៅក្នុង chapter នេះមាន:

- ការបើក files ដើម្បីបាន streams មកអនុវត្ត
- ការបិទ streams ដើម្បីរក្សាទិន្នន័យនៅក្នុងពួកវាទៅក្នុង files
- ការប្រតិបត្តិជាមួយនឹង text files
- ការប្រតិបត្តិជាមួយនឹង binary files
- ការប្រតិបត្តិជាមួយនឹង random access files

1. Introduction:

File គឺជាការប្រមូលផ្តុំ ឬសំនុំនៃតួអក្សរដែលផ្ទុកនៅក្នុង Physical Storage Devices ដូចជា Diskettes, Hard Disk ជាដើម។ នៅក្នុង C++, Files ត្រូវបានគេបែងចែកចេញជា ២ប្រភេទគឺ: Sequential files និង Random Access file។

Sequential files មានលក្ខណៈងាយស្រួលក្នុងការបង្កើតឡើងជាង Random Access files។ នៅក្នុង Sequential files, ទិន្នន័យ ឬ text នឹងត្រូវបានគេផ្ទុក ឬអានត្រលប់មកវិញតាមរយៈ បន្តបន្ទាប់គ្នាជារដឹក។ ចំណែកឯ Random Access files វិញ, ទិន្នន័យត្រូវបានគេ access និង process តាមលក្ខណៈចៃដន្យ។

នៅក្នុង Header file **fstream.h** បានផ្តល់អោយនូវការ process ទៅលើ **input** stream, **output** stream ។ Header file នេះមាន classes ចំនួនបីគឺ:

- **ifstream:** ប្រើសំរាប់អាន stream មួយនៃ object ចេញពី file ជាក់លាក់ណាមួយ
- **ofstream:** ប្រើសំរាប់សរសេរ stream មួយនៃ object ទៅលើ file ជាក់លាក់ណាមួយ
- **fstream:** ប្រើសំរាប់អាន និងសរសេរ stream មួយនៃ object ចេញពី និងទៅលើ file ជាក់លាក់ណាមួយ

2. Opening Files:

File មួយអាចត្រូវបានគេប្រើឡើងដើម្បីបាន stream របស់វាប្រើប្រាស់តាមរយៈ constructors ឬ member functions open() របស់ classes ទាំងបី: ifstream, ofstream, និង fstream ដូចបានរៀបរាប់

នៅផ្នែកខាងលើ។ Object របស់ classes ទាំងនេះពេលបើកបានជោគជ័យហើយ, វាគឺ streams តាងអោយ files ទាំងនោះ។ ទំរង់ទូទៅសំរាប់ការបើក files មានតាមរយៈ constructors គឺ:

- ifstream → ifstream (const char *filename, int mode=ios::in, int prot=ios::openprot);
- ofstream → ofstream (const char *filename, int mode=ios::out, int prot=ios::openprot);
- fstream → fstream(const char *filename, int mode, int prot=ios::openprot);

ដោយក្នុងនោះ ប៉ារ៉ាម៉ែត្រ

- file name គឺជាឈ្មោះរបស់ file (អាចរួមទាំង path ផងដែរ) ដែលនឹងត្រូវគេបើកដើម្បីបានជា stream សម្រាប់អានទិន្នន័យពីវា ឬសរសេរទិន្នន័យទៅកាន់វា។
- mode គឺជាចំនួនគត់ប្រើសំរាប់កំណត់ពី mode របស់ stream ដែលតាងអោយ file ដែលបានបើកឡើង។ modes របស់ streams មានតំលៃថេរដូចតារាងខាងក្រោម:

Mode	Meaning
ios::in	បើក file សម្រាប់អានទិន្នន័យ
ios::out	បើក file សម្រាប់សរសេរទិន្នន័យទៅកាន់
ios::app	បើក file សម្រាប់សរសេរទិន្នន័យភ្ជាប់បន្ត
ios::ate	បើក file ដោយមិនចាប់ផ្តើមចេញពីចំនុចចាប់ផ្តើម
ios::trunc	បើក file ដោយលប់ file ចោលបើ file នោះមានរួចហើយនិងជំនួសដោយ file ថ្មី
ios::nocreate	បើក file បាន បើសិនជា file មិនមានទេ
ios::noreplace	បើក file បើសិនជា file ពិតជាមាន
ios::binary	បើក file សម្រាប់ binary mode (ជា default គឺជា text mode)

- prot គឺជាចំនួនគត់កំនត់ពី access permission របស់ files។ វាត្រូវបានគេប្រើប្រាស់សម្រាប់តែករណីដែល mode របស់ stream ត្រូវបានគេកំណត់តំលៃជា ios::nocreate។ តំលៃថេរទាំងឡាយសម្រាប់ប៉ារ៉ាម៉ែត្រ prot មាននៅក្នុងតារាងខាងក្រោម:

Value	Meaning
0	Default
1	Read only file
2	Hidden file
4	System file
8	Archive file

នៅក្នុងករណីខ្លះ object របស់ classes: ifstream, ofstream និង fstream ត្រូវបានកកើតឡើងហើយ តែមិនទាន់តាងអោយ files នោះទេ (មានន័យថា ពួកវាកកើតឡើងតាម default constructors) ឬក៏គេត្រូវការអោយពួកវាកាយជា stream សម្រាប់ file ផ្សេងទៀត នោះ method: open() ត្រូវបានអនុវត្តសម្រាប់នេះ។ ទំរង់ទូទៅរបស់ member functions: open() របស់ classes ទាំងបីនោះគឺ:

- ifstream → void open (const char *filename, int mode=ios::in, int prot=ios::openprot);
- ofstream → void open (const char *filename, int mode=ios::out, int prot=ios::openprot);
- fstream → void open (const char *filename, int mode, int prot=ios::openprot);

ប៉ារ៉ាម៉ែត្រ: filename, mode និង prot របស់ method: open() ទាំងនេះមានអត្ថន័យដូចប៉ារ៉ាម៉ែត្ររបស់ constructors ដែរ។

ឧទាហរណ៍ទី១: បើក text file **myfile.xxx** មួយនៅក្នុង current directory សម្រាប់សរសេរទិន្នន័យទៅកាន់ stream របស់វា។

- ប្រើប្រាស់ constructor របស់ class **ofstream**

```
ofstream fout("myfile.xxx");
```

- ប្រើប្រាស់ constructor របស់ class **fstream**

```
fstream fout("myfile.xxx", ios::out);
```

- ប្រើប្រាស់ method open() របស់ ofstream object

```
ofstream fout;
fout.open("myfile.xxx");
```

- ប្រើប្រាស់ method open() របស់ fstream object

```
fstream fout;
fout.open("myfile.xxx", ios::out);
```

បន្ទាប់ពី statements នៃឧទាហរណ៍នេះត្រូវបានអនុវត្តហើយ បើ stream បានកើតឡើងដោយជោគជ័យ នោះគេបាន fout គឺជា output stream ដែលអនុញ្ញាតិអោយគេអាចសរសេរតួអក្សរទាំងឡាយទៅកាន់ stream បាន។

ឧទាហរណ៍ទី២: បើក text file **myfile.xxx** មួយនៅក្នុង **root** directory នៃ Drive C: សម្រាប់អានទិន្នន័យចេញ។

- ប្រើប្រាស់ constructor របស់ class **ifstream**

```
ifstream fin("c:\\myfile.xxx");
```

- ប្រើប្រាស់ constructor របស់ class **fstream**

```
fstream fin("c:\\myfile.xxx", ios::in);
```

- ប្រើប្រាស់ method open() របស់ ifstream object

```
ifstream fin;
fin.open("c:\\myfile.xxx");
```

- ប្រើប្រាស់ method open() របស់ fstream object

```
fstream fin;
fin.open("c:\\myfile.xxx", ios::in);
```

បន្ទាប់ពី statements នៃឧទាហរណ៍នេះត្រូវបានអនុវត្តហើយ បើ stream បានកើតឡើងដោយជោគជ័យ នោះគេបាន fout គឺជា input stream ដែលអនុញ្ញាតិអោយគេអាចអានតួអក្សរទាំងឡាយទៅកាន់វាបាន។

ឧទាហរណ៍ទី៣: បើក text file myfile.xxx មួយនៅក្នុង directory c:\tc\ នៃ Drive C: សម្រាប់អានទិន្នន័យចេញផង និងសរសេរទិន្នន័យទៅកាន់ stream របស់វាផង។

- ប្រើប្រាស់ constructor របស់ class **fstream**

```
fstream finout("c:\tc\myfile.xxx", ios::in | ios::out);
```

- ប្រើប្រាស់ method **open()** របស់ fstream object

```
fstream finout;
finout.open("c:\tc\myfile.xxx", ios::in | ios::out);
```

បន្ទាប់ពី statements នៃឧទាហរណ៍នេះត្រូវបានអនុវត្តហើយ បើ stream បានកើតឡើងដោយជោគជ័យ នោះគេបាន object finout តាងអោយ stream របស់ file c:\tc\myfile.xxx និង input-output stream អនុវត្តអានទិន្នន័យចេញ និងសរសេរទិន្នន័យទៅកាន់ stream។

ឧទាហរណ៍ទី៤: បើក text file **myfile.xxx** មួយនៅក្នុង **current** directory សម្រាប់សរសេរទិន្នន័យភ្ជាប់បន្ត។

- ប្រើប្រាស់ constructor របស់ class **ofstream**

```
ofstream fout("myfile.xxx", ios::app);
```

- ប្រើប្រាស់ constructor របស់ class **fstream**

```
fstream fout("myfile.xxx", ios::out | ios::app);
```

- ប្រើប្រាស់ method **open()** របស់ ofstream object

```
ofstream fout;
fout.open("myfile.xxx");
```

- ប្រើប្រាស់ method **open()** របស់ fstream object

```
fstream fout;
fout.open("myfile.xxx", ios::out | iso::app);
```

3. Closing Files

ការបិទ file មួយដែលបានបើករួចហើយមានប្រយោជន៍ដើម្បីរក្សាទិន្នន័យនៅលើ stream របស់ file ទៅកាន់ file នៅក្នុង physical storage devices។ សម្រាប់ streams របស់ files ទាំងឡាយដែលគេលែងត្រូវការអាន ឬសរសេរទិន្នន័យ គេគួរតែបិទវាដើម្បី release memory space ដែលប្រើប្រាស់ដោយ streams ទាំងនោះ។ ដើម្បីធ្វើការបិទ file មួយគេប្រើ function member: close() របស់ classes ទាំងបី: fstream, ifstream, ofstream។ method close() នេះមិនត្រូវការ argument ទេ។

ឧទាហរណ៍ទី៥: បិទ streams របស់ file ដែលគេបានបើក។

```
fstream finout("myfile.txt", ios::in | ios::out);
ifstream fin;
.........
.........
fin.open("youfile.txt");
.........
.........
finout.close();
.........
.........
fin.close();
```

4. Stream State Member functions

នៅក្នុងភាសា C++, file stream classes មាន state member functions សម្រាប់ផ្តល់ពត៌មានស្តីពីស្ថានភាពរបស់ stream ដូចជា ការអានបានឈានមកដល់ទីបញ្ចប់របស់ file ហើយ ឬនៅ, file ត្រូវបានបើកឡើងដោយជោគជ័យ ឬយ៉ាងណាជាដើម។ល។ member functions បញ្ជាក់ពីស្ថានភាពរបស់ stream មានដូចជា: eof(), fail(), good(), bad()។

eof() ជា stream state function ដែលត្រូវបានគេប្រើប្រាស់សម្រាប់ពិនិត្យមើលថាតើ file pointer បានឈានមកដល់ទីបញ្ចប់របស់តួអក្សរចុងក្រោយហើយ ឬនៅ។ បើសិនជា file pointer មិនទាន់បានឈានមកដល់ទីបញ្ចប់របស់តួអក្សរចុងក្រោយនោះទេវា return តំលៃ 0 បើពុំដូច្នោះទេវា return តំលៃខុសពី 0។

ឧទាហរណ៍ទី៦: ប្រើ method eof() សម្រាប់ការបញ្ចប់ការអានទិន្នន័យចេញពី stream របស់ file។

```
ifstream fin("myfile.txt");
.........
.........
while(1){
     // read a character from stream fin here
     if (fin.eof()) break;
     // do something here with the character read out
}
fin.close();
```

fail() ជា stream state function ដែលត្រូវបានគេប្រើប្រាស់សម្រាប់ពិនិត្យមើលថាតើ file បានបើកឡើងសម្រាប់អាន ឬសរសេរនោះបានជោគជ័យរឺអត់? ឬការព្យាយាមអនុវត្ត invalid operations ។ បើសិនជាបរាជ័យវានឹង return តំលៃខុសពី 0, បើពុំដូច្នោះទេវា return តំលៃ 0។

ឧទាហរណ៍ទី៧: ប្រើ method fail() សម្រាប់ធានាការអនុវត្ត input-output operation ទៅលើតែ stream ដែលគេបានបើកឡើងបានដោយជោគជ័យ។

```
ifstream fin("myfile.txt");
if(fin.fail()){
     cout << "Error in opening file for reading\n";
     return;
}
while(1){
     // read a character from stream fin here
     if (fin.eof()) break;
     // do something here with the character read out
}
fin.close();
```

good() ជា stream state function ដែលត្រូវបានគេប្រើប្រាស់សម្រាប់ពិនិត្យមើលថាតើ previous file operation អនុវត្តបានជោគជ័យដែរឬទេ? បើសិនជាបានជោគជ័យ វានឹង return តំលៃខុសពី 0, បើពុំដូច្នោះទេវា return តំលៃ 0។

ឧទាហរណ៍ទី៨: ប្រើ method good() សម្រាប់ពិនិត្យ valid file operation ។

```
fstream f("myfile.txt", ios::in);
if(f.fail()){
     cout << "Error in opening file for reading\n";
     return;
}
.........
// reading a character from stream f here
if (!f.good()){
     cout << "Failure in reading a character from file\n";
}
.........
fin.close();
```

bad() ជា stream state function ដែលត្រូវបានគេប្រើប្រាស់សម្រាប់ពិនិត្យមើលថាតើ invalid file operations ត្រូវបានគេព្យាយាមអនុវត្តជាមួយនឹង stream, ឬមាន error កើតឡើងចំពោះការអានទិន្នន័យដែរឬទេ? បើសិនជាមានរឿងនេះកើតឡើង នោះវានឹង return តម្លៃខុសពី 0, បើពុំដូច្នោះទេវា return តម្លៃ 0។

ឧទាហរណ៍ទី៩: ប្រើ method bed() សម្រាប់ពិនិត្យ invalid file operation ។

```
fstream f("myfile.txt", ios::in);
```

```
if(f.fail()){
    cout << "Error in opening file for reading\n";
    return;
}
.........
// writing a string to stream f here
if (f.bad()){
    cout << "Writing to file is not allowed\n";
}
.........
f.close();
```

5. Text File Operations

ការសរសេរទិន្នន័យទៅកាន់ text file គឺជាការ output តួអក្សរទាំងឡាយទៅកាន់ stream តាងអោយ file ដែលគេបាន open។ បើសិនជាទិន្នន័យដែលត្រូវសរសេរទៅកាន់មានប្រភេទ មិនមែនជាតួអក្សរ ទិន្នន័យនោះត្រូវបំលែងជា string សមមូល រួចទើបផ្ទុកនៅក្នុង file stream ជាតួអក្សរទាំងឡាយ។ ទិន្នន័យដែលបានសរសេរទៅកាន់ stream ត្រូវបានរក្សាទុកក្នុង file នៅពេល stream តាងអោយ file នោះត្រូវបានបិទ។ តួអក្សរទាំងឡាយត្រូវសរសេរទៅកាន់ stream របស់ file នៅត្រង់ទីតាំងរបស់ current file pointer, ហើយក្រោយ operation បានបញ្ចប់ file pointer បានរំកិលទៅមុខត្រង់ទីតាំងបន្ទាប់ពីតួអក្សរចុងក្រោយដែលបានសរសេរចូលទៅ។ សំរាប់ stream ប្រភេទជា text គេអនុវត្តជាមួយនឹងប្រមាណវិធី insertion (<<), method put() សំរាប់សរសេរទិន្នន័យទៅកាន់ output stream ដូចដែលធ្លាប់អនុវត្តជាមួយនឹង object cout សំរាប់សរសេរទិន្នន័យទៅកាន់ standard output ដែរ។

ឧទាហរណ៍ទី១0: Insertion operator, <<, ប្រើសំរាប់សរសេរទិន្នន័យទៅ stream របស់ text file។

```
ofstream fout("myfile.txt");
if(fout.fail()){
    cout << "Error in opening file for writing" << endl;
    return;
}

int x = 23, y = 89;
fout << "File Operation" << endl;
fout << "x : " << x << endl;
fout << "y : " << y << endl;
fout << "Sum is "<< (x+y) << endl;
fout.close();
```

កម្មវិធីទី១: កម្មវិធីសំរាប់សរសេរតួអក្សរទាំងឡាយទៅកាន់ stream របស់ text file។

```
#include <fstream.h>
#include <iostream.h>
#include <conio.h>

void main() {
```

```
    clrscr();
    fstream fout("myfile.txt");
    if(fout.fail()){
        cout << "Error in opening file for writing" << endl;
        getch();
        return;
    }

    fout << "1- Demonstrating of text file processing" << endl;
    fout << "2- Welcome to Cambodia" << endl;
    fout << "3- Kingdom of Cambodia" << endl;
    fout << "4- Nation Religion King" << endl;
    fout << "5- Bye..." << endl;
    fout.close();
    cout << "Five Line of text were successfully written "
         << "to file myfile.txt" << endl;
    getch();
}
```

លទ្ធផលរបស់កម្មវិធីខាងលើ បើសិនជាបើក text file បានដោយជោគជយ័ វាបង្កើតបានជា file myfile.txt មានទីតាំងនៅក្នុង current directory ហើយ file នេះមាន stream តាងអោយវាជា **fout**។ តាមរយៈ stream fout នេះ កម្មវិធីនឹងសរសេរ text ដែលមាន ៥បន្ទាត់ទៅកាន់ខ្លួនវា បន្ទាប់មកត្រូវរក្សាទុកនៅក្នុង stream នេះទៅកាន់ file myfile.txt។ ចំពោះករណីនៅក្នុង current directory មាន file myfile.txt រួចហើយ នោះវានឹងលុប file ចាស់ចោល ហើយបង្កើត file ថ្មីជំនួសវិញ។ ជាវិបាកទិន្នន័យដែលមាននៅក្នុង file ចាស់មិនអាចស្រោចស្រង់បានវិញទេ លើកលែងតែគេអនុវត្តការ backup ឬមួយក៏ត្រូវសង្គ្រោះមកវិញតាមកម្មវិធីផ្សេងទៀត។

កម្មវិធីទី២: កម្មវិធីប្រើប្រាស់ text file សំរាប់ផ្ទុក text ដែលបានមកពីការសរសេរទៅជាតួអក្សរទាំងឡាយ ចំនួនគត់ទាំងឡាយ ចំនួនទសភាគទាំងឡាយ ដោយមានកំណត់ទំរង់បង្ហាញចេញរបស់ចំនួនទសភាគអោយមាន ២ខ្ទង់នៅខាងក្រោយក្បៀស។

```
#include <fstream.h>
#include <iostream.h>
#include <conio.h>

void main() {

    char* fileName = "myfile1.txt";
    fstream fout(fileName, ios::out);

    if(fout.fail()){
        cout << "Error in opening file " << fileName << endl;
        getch();
        return;
    }

    int n = 0;
```

```
    float a, b, r;
    char ch;

    cout.setf(ios::fixed|ios::showpoint); cout.precision(2);

    fout.setf(ios::fixed|ios::showpoint); fout.precision(2);

    do {
        cout << "Operation " << (n+1) << endl;
        cout << "Input a, b: "; cin >> a >> b;
        r = a + b;

        cout << "a = " << a << endl;
        cout << "b = " << b << endl;
        cout << "a + b = " << r << endl;

        fout << "Operation " << (n+1) << endl;
        fout << "a = " << a << endl;
        fout << "b = " << b << endl;
        fout << "a + b = " << r << endl;

        n++;
        cout << "\nOne more operation? [Yes/No]: ";
        ch = getche();

        cout << endl;
        fout << endl;

    }while(ch=='y' || ch=='Y');

    fout.close();
}
```

កម្មវិធីទី៣: កម្មវិធីសំរាប់សរសេរទិន្នន័យរបស់ objects មួយទៅកាន់ text file។

```
#include <fstream.h>
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

class Student{
    public:
        char fullName[30];
        char gender[7];
        int age;
        char address[50];
};

void main(){

    Student stu;
    char fileName[100];
    char ch;

    cout << "Enter the file name that you want to create: ";
    gets(fileName);
```

```
    clrscr();

    fstream fout(fileName, ios::nocreate);

    if(fout.fail()) {                       // file does not exist
        fout.open(fileName, ios::app);      // for appending
        if(fout.fail()) {
            cout << "Error in opening file " << fileName << endl;
            getch();
            return;
        }
    }

    gotoxy(1, 1); cout << "N";
    gotoxy(6, 1); cout << "Full Name";
    gotoxy(30, 1); cout << "Gender";
    gotoxy(40, 1); cout << "Age";
    gotoxy(50, 1); cout << "Address";

    int i = 1, y = 2;
    do {
        clreol();

        gotoxy(1, y); cout << i;
        gotoxy(6, y); gets(stu.fullName);
        gotoxy(30, y); gets(stu.gender);
        gotoxy(40, y); cin >> stu.age;
        gotoxy(50, y); gets(stu.address);

        fout << i << "   " << stu.fullName << "   " << stu.gender << "   "
             << stu.age << "   " << stu.address;

        gotoxy(1, 25); cout << "Do you want to continue? [Yes/No]: ";
        ch = getche();
        if(ch=='y' || ch=='Y')
            gotoxy(1, 25); clreol();
        y++; i++;
    }while(ch=='y' || ch=='Y');

    fout.close();
    cout << "\nCompleted" << endl;
    cout << "Press any key to exit..." << endl;
    getch();
}
```

ឧទាហរណ៍ទី១១: អានយកតួអក្សរចាប់ពីទីតាំងរបស់ current file pointer រហូតដល់តួអក្សរចុងក្រោយចេញពី input stream **fin** របស់ text file មកបង្ហាញនៅលើ screen។

```
    .........
    .........
    char ch;
    while(1) {
        ch = fin.get()
        if(fin.eof()) break;
```

```
        cout << ch;
    }
    f.close();
```

Method get() អនុវត្តអានយកមួយតួអក្សរចេញពី input stream ដោយមិនប្រកាន់ថាតួអក្សរត្រង់ទីតាំងរបស់ current file pointer ថាជាតួអក្សរអ្វីនោះទេ ចំនែកឯប្រមាណវិធី extraction ដែលត្រូវបានយកទៅអនុវត្តនៅក្នុងការអានអក្សរមួយ ឬតួអក្សរច្រើនចេញពី input stream ទៅអោយអញ្ញាតិផ្សេងៗនោះ វានឹងមិនទទួលយក និងបញ្ចប់ការអានបន្ទាប់ពីបានអានតួអក្សរជា space ឬជាតួអក្សរចុះបន្ទាត់។

ឧទាហរណ៍ទី១២: អានតួអក្សរទាំងអស់លើកលែងតែ space និងតួអក្សរចុះបន្ទាត់ ចាប់ពីទីតាំងរបស់ current file pointer រហូតទៅដល់តួអក្សរចុងក្រោយចេញពី input stream **fin** រួចសរសេរទៅកាន់ output stream fout។

```
.........
.........
char ch;
while(1) {
    fin >> ch;
    if(fin.eof()) break;
    fout << ch;
}
fin.close();
fout.close();
```

ឧទាហរណ៍ទី១៣: អានចំនួនគត់ (int), string, ចំនួនទសភាគ (float) បន្តគ្នាចេញពី input stream របស់ text file ឈ្មោះ data.txt នៅក្នុង current directory ដែលផ្ទុកទិន្នន័យតែមួយបន្ទាត់មាន ចំនួនគត់, space, string, ចំនួនទសភាគ។ ឧទាហរណ៍ (13 Monypiseth 89.50)

```
fstream fin("data.txt", ios::in);
if(fin.fail()) {
    cout << "Error for opening file for reading." << endl;
} else {
    int id;
    float gpa;
    char name[25];

    fin >> id;
    fin >> ws >> name;
    fin >> gpa;
    fin.close();
    .........
    .........
}
```

កម្មវិធីទី៤: អាន contents ទាំងអស់របស់ text file myfile.txt បង្កើតឡើងតាម កម្មវិធីទី១ មកបង្ហាញនៅលើ screen។

```
#include <fstream.h>
#include <iostream.h>
#include <conio.h>

void main() {

     ifstream fin("myfile.txt");

     if(fin.fail()){
          cout << "Error in opening file for reading" << endl;
          getch();
          return;
     }

     char ch;
     cout << "Here is the contents of text file myfile.txt: " << endl;

     while(1) {
          ch = fin.get();
          if(fin.eof()) break;
          cout << ch;
     }

     fin.close();
     getch();
}
```

កម្មវិធីទី៥: ចំលង contents ទាំងអស់របស់ text file myfile.txt បង្កើតឡើងតាម កម្មវិធីទី១ មកដាក់ក្នុង file ថ្មីមួយ ឈ្មោះថា myfile1.txt ។

```
#include <fstream.h>
#include <iostream.h>
#include <conio.h>

void main() {

     ifstream fin("myfile.txt");

     if(fin.fail()){
          cout << "Error in opening file for reading" << endl;
          getch();
          return;
     }

     ofstream fout("mydata.txt");

     if(fout.fail()){
          cout << "Error in opening file for writing" << endl;
          getch();
          return;
```

```
    }

    char ch;
    cout << "Here is the contents of text file myfile.txt: " << endl;

    while(1) {
        ch = fin.get();
        if(fin.eof()) break;
        fout << ch;
    }

    fin.close();
    fout.close();
    cout << "Contents of myfile.txt was copied to the new file mydata.txt"
         << endl;
    getch();
}
```

កម្មវិធីទី៥៖ កម្មវិធីនេះបញ្ជាក់ពីការអាន strings ចំនួនទសភាគទាំងឡាយចេញពី text file myfile1.txt បង្កើតឡើងតាម កម្មវិធីទី២។

```
#include <fstream.h>
#include <iostream.h>
#include <conio.h>

void main() {

    ifstream fin("myfile1.txt");

    if(fin.fail()){
        cout << "Error in opening file for reading" << endl;
        getch();
        return;
    }

    float a, b;
    char s[100], awayStr[100];
    cout.setf(ios::fixed | ios::showpoint); cout.precision(3);

    while(1) {
        fin.get(s, 100);
        // read to the next line
        while(fin.get()!='\n');
        if(fin.eof()) break;

        fin.get(awayStr, 100, '=');   // read character before '='
        fin.get();                    // read character '=' away
        fin >> a;

        // read to the next line
        while(fin.get()!='\n');
        if(fin.eof()) break;

        fin.get(awayStr, 100, '=');   // read character before '='
        fin.get();                    // read character '=' away
```

```
        fin >> b;

        // read to the next line
        while(fin.get()!='\n');
        if(fin.eof()) break;

        cout << s << endl;
        cout << "a = "; cout.width(10); cout << a << endl;
        cout << "b = "; cout.width(10); cout << a << endl;
        cout << "a+b = "; cout.width(10); cout << (a+b) << endl;
        cout << endl;

        // read to the next line
        while(fin.get()!='\n');
        if(fin.eof()) break;

        // skip a blank line
        fin >> ws;
        if(fin.eof()) break;
    }

    fin.close();
    getch();
}
```