អំពី **Templates** និង **Exceptions**

ជំពូកទី ១២

១. អំពី Function Template

Template គឺជាវិធីមួយសម្រាប់សរសេរអនុគមន៍ ឬ class តាមប្រភេទអនុគមន៍ឬ class ដែលមានលក្ខណៈស្រដៀងគ្នានិងមានលក្ខណៈរួមមួយជាទូទៅ។ នៅពេលអនុគមន៍រួមមួយត្រូវបានសរសេរឡើង ដើម្បីបានប្រភេទអនុគមន៍ដែលមានលក្ខណៈប្រហាក់ប្រហែលគ្នានោះ​គេហៅថា "function template"។ អនុគមន៍នេះមាន formal argument មួយយ៉ាងតិចដែលជាលក្ខណៈទូទៅ។

នៅក្នុងជំពូកមុនៗនេះ គេបានពន្យល់រួចហើយអំពីអនុគមន៍មួយអាចធ្វើ overloaded សម្រាប់ការប្រតិបត្តិការដែលមានលក្ខណៈប្រហាក់ប្រហែលគ្នាដើម្បីធ្វើការប្រតិបត្តិ ហើយនិងរបៀបបង្កើតអនុគមន៍ទាំងនេះ ព្រមទាំងការហៅមកប្រើនៅក្នុងកម្មវិធីមួយ។ នៅក្នុង function overloading ទោះជាវាមានឈ្មោះដូចគ្នាហើយត្រូវបានប្រើសម្រាប់អនុគមន៍ទាំងអស់ដែលបានបង្កើតតាមលក្ខណៈ overloaded ក៏ដោយ ប៉ុន្តែ codes មានលក្ខណៈច្រំដែលចំពោះគ្រប់ អនុគមន៍ទាំងអស់។ វាមានតែឈ្មោះអនុគមន៍ប៉ុណ្ណោះដែលដូចគ្នា ប៉ុន្តែការប្រកាសអនុគមន៍និង function definition មាន លក្ខណៈច្រំដែល។

ឧទាហរណ៍ ៖

**swap(char \*, char \*){ //** ប្តូរពីរតម្លៃជាប្រភេទ char

**.....**

**.....**

**}**

**swap(int , int){ //** ប្តូរពីរតម្លៃជាប្រភេទ int

**.....**

**.....**

**}**

**swap(float, float){ //** ប្តូរពីរតម្លៃជាប្រភេទ float

**.....**

**.....**

**}**

C++ ផ្តល់នូវលក្ខណៈពិសេសដែលអាចឲ្យគេបង្កើតអនុគមន៍រួមមួយសម្រាប់ សំណុំ អនុគមន៍ដែលមានលក្ខណៈប្រហាក់ប្រហែលគ្នា។ កាលណា អនុគមន៍រួមបានសរសេរឡើងសម្រាប់ពពួកអនុគមន៍ដែលមានលក្ខណៈប្រហាក់ប្រហែលគ្នានោះ វាត្រូវបានហៅថា function templates។ សារៈសំខាន់នៃការប្រើ function template គឺជៀសវាងនូវការប្រើ source code ច្រំដែល។ code នៃ object បានធ្វើឲ្យមានឥទ្ធិពល និងមានប្រសិទ្ធភាពច្រើនជាងការប្រើវិធី ធម្មតានៃការប្រកាស និងការបង្កើតអនុគមន៍។ សារៈសំខាន់នៃការប្រើ function template ទីពីរ គឺការផ្ទៀងផ្ទាត់ប្រភេទទិន្នន័យត្រូវបានអនុវត្តយ៉ាងពេញលេញ។ function template ពុំបានបញ្ជាក់នូវប្រភេទទិន្នន័យរបស់ argument ដែលអនុគមន៍បានទទួលយកឡើយ តែវាប្រើប្រភេទទិន្នន័យជាប៉ារ៉ាម៉ែត្រ ឬជាលក្ខណៈទូទៅ។ នៅក្នុង function template យ៉ាងហោចណាស់ក៏មាន formal argument មួយដែរដែលជាលក្ខណៈទូទៅ។

ទំរង់ទូទៅនៃការប្រកាស function template មួយតាមភាសា C++ គឺ ៖

*template* <class T> T *function\_name* (T *formal arguments*) {

. . . .

. . . .

*return* (T);

}

ដែលក្នុងនេះ template និង class គឺជា keywords របស់ C++ ហើយ function template ត្រូវតែ ចាប់ផ្តើមដោយប្រើពាក្យ template និង T គឺជាប្រភេទទិន្នន័យមានលក្ខណៈប៉ារ៉ាម៉ែត្រ។

ការប្រកាស function template ខាងលើនេះអាចសរសេរជាទម្រង់មួយទៀតដូចខាងក្រោមនេះ  គឺ៖

*template* <class T>

T *function\_name* (T *formal arguments*){

. . . .

. . . .

*return* (T);

}

អ្នកប្រើប្រាស់ប្រហែលជាមានការអន្ទះសាចង់ដឹងពីប្រភេទ argument ដែលបានកំណត់នោះហើយ។ return type របស់អនុគមន៍មិនដែលគិតឡើយជាមួយនឹងប្រភេទទិន្នន័យរបស់ប៉ារ៉ាម៉ែត្រសម្រាប់ដំណើរការ។ ប្រភេទទិន្នន័យនៃអនុគមន៍ត្រូវតែស៊ីសង្វាក់គ្នាជាមួយនឹង formal arguments នៃការប្រកាសអនុគមន៍ ទោះជា arguments មានលក្ខណៈជាប៉ារ៉ាម៉ែត្រត្រូវបានប្រមូលផ្តុំទៅក្នុង function template ក៏ដោយ។

ឧទាហរណ៍នៃការប្រកាស function template មួយចំនួនមានដូចខាងក្រោម ៖

 function template ត្រូវបង្កើតឡើងសម្រាប់គណនាផលបូកនៃធាតុរបស់ array ដែលគេឲ្យជា int, float ឬ double -ល-។

**template <class T>**

**T sum (T \*array, int n){ //** ប៉ារ៉ាម៉ែតទូទៅ

**T temp = 0; //** ការប្រកាសអញ្ញាត temp ជាប្រភេទ T ដោយកំណត់តម្លៃស្មើសូន្យ

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

**temp = temp + array[i]; //** ធ្វើប្រមាណវិធីបូកតម្លៃទាំងអស់ក្នុង array រួច

**//** កំណត់តម្លៃទៅឲ្យអញ្ញាត temp

**return (temp);**

**}**

 function template ត្រូវបង្កើតឡើងសម្រាប់ធ្វើការប្តូររបស់ (items) ពីរដែលមានប្រភេទទិន្នន័យផ្សេងគ្នា ដូចជា int, float, double ឬ ជាតួអក្សរ។

**template <class T>**

**T swap (T &first, T &second){**

**T temp; //** ការប្រកាសអញ្ញាត temp ជាប្រភេទ T

**temp = first;**  **//** កំណត់តម្លៃអញ្ញាត first ទៅឲ្យអញ្ញាត temp

**first = second; //** កំណត់តម្លៃអញ្ញាត second ទៅឲ្យអញ្ញាត first

**second = temp; //** កំណត់តម្លៃអញ្ញាត second ទៅឲ្យអញ្ញាត temp

**return (0);**

**}**

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីការបង្កើត function template មួយសម្រាប់គណនា ផលបូក array នៃចំនួនគត់ និង array នៃចំនួនទសភាគ។

**// using function template**

**#include <iostream.h>**

**template<class T> T sum (T \*array, int n){**

**T temp = 0; //** ការប្រកាសអញ្ញាត temp ជាប្រភេទ T ដោយកំណត់តម្លៃស្មើសូន្យ

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

**temp = temp + array[i]; //** ធ្វើប្រមាណវិធីបូកតម្លៃទាំងអស់ក្នុង array រួច

**//** កំណត់តម្លៃទៅឲ្យអញ្ញាត temp

**return temp;**

**}**

**void main(){**

**int n = 3, sum1; //** ការប្រកាសអញ្ញាត n, sum1 ជាប្រភេទ int

**float sum2; //** ការប្រកាសអញ្ញាត sum2 ជាប្រភេទ float

**static int a[3] = {1, 2, 3}; //** ការប្រកាស array a ជាប្រភេទ int និង static

**static float b[3] = {1.1, 2.2, 3.3 };**

**sum1 = sum(a, n); //** ការហៅអនុគមន៍ sum() មកប្រើ រួចកំណត់តម្លៃឲ្យអញ្ញាត sum1

**cout << "Sum of the integer = " << sum1 << endl;**

**sum2 = sum(b, n); //** ការហៅអនុគមន៍ sum() មកប្រើ រួចកំណត់តម្លៃឲ្យអញ្ញាត sum2

**cout <<"Sum of the floating point numbers = "<< sum2;**

**cout << endl;**

**}**

កម្មវិធីខាងលើនេះ function template ឈ្មោះ sum() ត្រូវបានបង្កើតឡើងជា អនុគមន៍ទូទៅសម្រាប់ធ្វើផលបូក array នៃតម្លៃចំនួន n ធាតុដែលចំនួនធាតុទាំងនេះត្រូវបញ្ជូនតាម រយៈការហៅអនុគមន៍មកប្រើ ពីផ្នែកណាមួយនៃកម្មវិធី។ function template ឈ្មោះ sum() ត្រូវបានហៅមកប្រើចំនួនពីរដង គឺ លើកទីមួយគណនាផលបូកនៃ array ជាចំនួនគត់ និងលើកទីពីរ គណនាផលបូកនៃ array ជាចំនួនទសភាគ។

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីការបង្កើត function template មួយ ដើម្បីធ្វើការប្តូរ items ពីរដែលមានប្រភេទទិន្នន័យផ្សេងគ្នា ដូចជាចំនួនគត់ និងចំនួនទសភាគ។

**// using function template**

**#include <iostream.h>**

**template<class T> T swap (T &first, T &second){**

**​ T temp = 0; //** ការប្រកាសអញ្ញាត temp ជាប្រភេទ T ដោយកំណត់តម្លៃស្មើសូន្យ

**temp = first; //** កំណត់តម្លៃអញ្ញាត first ទៅឲ្យអញ្ញាត temp

**first = second; //** កំណត់តម្លៃអញ្ញាត second ទៅឲ្យអញ្ញាត first

**second = temp; //** កំណត់តម្លៃអញ្ញាត second ទៅឲ្យអញ្ញាត temp

**return 0;**

**}**

**void main(){**

**int ix, iy; //** ការប្រកាសអញ្ញាត ix, iy ជាប្រភេទ int

**float fx, fy; //** ការប្រកាសអញ្ញាត fx, fy ជាប្រភេទ float

**cout << "Enter any two integers :\n";**

**cin >> ix >> iy; //** បញ្ចូលតម្លៃឲ្យអញ្ញាត ix, និង iy តាមរយៈ keyboard

**cout << "Enter any two floating point numbers?:\n";**

**cin >> fx >> fy; //** បញ្ចូលតម្លៃឲ្យអញ្ញាត fx, និង fy តាមរយៈ keyboard

**swap(ix, iy); //** ការហៅអនុគមន៍ swap() មកប្រើ

**cout << "After swapping integers :\n";**

**cout << "ix = " << ix << " iy = " << iy << endl;**

**swap(fx, fy); //** ការហៅអនុគមន៍ swap() មកប្រើ

**cout << "After swapping floating point numbers :\n";**

**cout << "fx = " << fx << " fy = " << fy << endl;**

**}**

ឧទាហរណ៍ទី ៣ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញពីការបង្កើត function template មួយឈ្មោះ square() ដើម្បីរកការេនៃចំនួនដែលគេឲ្យ ដោយមានប្រភេទទិន្នន័យផ្សេងគ្នាដូចជា ចំនួនគត់, float និង double។

**// using function template**

**#include <iostream.h>**

**template<class T> T square (T one){**

**T one;**

**return (one \* one);**

**}**

**void main(){**

**int x, xsq; //** ការប្រកាសអញ្ញាត x, xsq ជាប្រភេទ int

**float y, ysq; //** ការប្រកាសអញ្ញាត y, ysq ជាប្រភេទ float

**double z, zsq; //** ការប្រកាសអញ្ញាត z, zsq ជាប្រភេទ double

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

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

**cout << "Enter a floating point number :\n";**

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

**cout << "Enter a double precision number :\n";**

**cin >> z; //** បញ្ចូលតម្លៃឲ្យអញ្ញាត z តាមរយៈ keyboard

**xsq = square(x); //** ការហៅអនុគមន៍ square() មកប្រើ

**cout <<"x = "<< x<<" and its square = "<< xsq<< endl;**

**ysq = square(y); //** ការហៅអនុគមន៍ square() មកប្រើ

**cout <<"y = "<< y<<" and its square = "<< ysq<< endl;**

**zsq = square(z); //** ការហៅអនុគមន៍ square() មកប្រើ

**cout <<"z = "<< z<<" and its square = "<< zsq<< endl;**

**}**

២. អំពី Class Template

ក្រៅពី function template នេះ C++ នៅមានការផ្តល់លក្ខណៈជា class template ដែរ។ តាមនិយមន័យ class template គឺជាការបង្កើត class មួយដែលរៀបរាប់ពីពពួក class មានទំនាក់ទំនងគ្នា។ C++ អាចឲ្យអ្នកប្រើប្រាស់មានលទ្ធភាពបង្កើត class មួយដែលមានប្រភេទទិន្នន័យមួយឬច្រើនប្រភេទ ដោយមានលក្ខណៈទូទៅឬលក្ខណៈជាប៉ារ៉ាម៉ែត្រ។ បែបបទនៃការប្រកាស class template គឺដូចគ្នាទៅនឹង function template ដែរ។ ពាក្យ template ត្រូវតែដាក់មុនគេនៃការបង្កើត class template។

ទំរង់ទូទៅនៃការបង្កើត class template គឺ ៖

*template* <class T>

class *user\_defined\_name* {

*private* :

. . . .

. . . .

*public* :

. . . .

. . . .

};

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត និងប្រកាស class template មួយជាភាសា C++។

**#include <iostream.h>**

**template <class T>**

**class sample {**

**private :**

**T value, value1, value2;**

**//** ការប្រកាសអញ្ញាត value, value1, value2 ជាប្រភេទ T មានលក្ខណៈ private

**public :**

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

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

**};**

**void main(){**

**sample <int> obj1; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

**sample <float> obj2; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj2

**....**

**....**

**}**

កាលណា class template បានបង្កើតឡើងនោះវាត្រូវការបង្កើត class object ដោយប្រើប្រភេទទិន្នន័យកំណត់ដោយអ្នកប្រើប្រាស់ ឬប្រភេទទិន្នន័យធម្មតាទៅជំនួសឲ្យប្រភេទទិន្នន័យជាប៉ារ៉ាម៉ែត្រ។

Member function នៃ class template មានប្រើពាក្យ template ផងដែរ ទោះជាវាប្រកាសនៅខាងក្រៅ scope នៃការបង្កើត class ក៏ដោយ។

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញនូវរបៀបប្រកាស member function សម្រាប់ class template មួយ។

**#include <iostream.h>**

**template <class T>**

**class sample {**

**private :**

**T value, value1, value2;**

**//** ការប្រកាសអញ្ញាត value, value1, value2 ជាប្រភេទ T មានលក្ខណៈ private

**public :**

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

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

**};**

**template <class T>**

**void sample <T>::getdata(){**

**cin >> value1 >> value2;**

**//** បញ្ចូលតម្លៃឲ្យអញ្ញាត value1, value2 តាមរយៈ keyboard

**}**

**template <class T>**

**void sample <T> :: sum(){**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T

**value = value1 + value2; //** ធ្វើផលបូក២តម្លៃរួចកំណត់ទៅឲ្យអញ្ញាត value

**cout << "Sum of = " << value << endl;**

**}**

ឧទាហរណ៍ទី ១ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញរបៀបបង្កើត និងប្រកាស class template ដើម្បីអានបញ្ចូល data items ចំនួនពីរពី keyboard ហើយនិងរកផលបូកនៃពីរ data items ដែលគេឲ្យ។

**// adding two parameterized data types**

**#include <iostream.h>**

**template <class T>**

**class sample {**

**private :**

**T value, value1, value2;**

**//** ការប្រកាសអញ្ញាត value, value1, value2 ជាប្រភេទ T មានលក្ខណៈ private

**public :**

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

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

**};**

**template <class T>**

**void sample <T>::getdata(){**

**​ cin >> value1 >> value2;**

**//** បញ្ចូលតម្លៃឲ្យអញ្ញាត value1, value2 តាមរយៈ keyboard

**}**

**template <class T>**

**void sample <T>::sum(){**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T

**value = value1 + value2; //** ធ្វើផលបូក២តម្លៃរួចកំណត់ទៅឲ្យអញ្ញាត value

**cout << "Sum of = " << value << endl;**

**}**

**void main(){**

**sample <int> obj1; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

**sample <float> obj2; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj2

**cout << "Enter any two integers :" << endl;**

**obj1.getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ obj1 នៃ sample class

**obj1.sum(); //** ការហៅអនុគមន៍ sum() តាមរយៈ obj1 នៃ sample class

**cout <<"Enter any two floating point numbers:"<<endl;**

**obj2.getdata(); //** ការហៅអនុគមន៍ getdata() តាមរយៈ obj2 នៃ sample class

**obj2.sum(); //** ការហៅអនុគមន៍ sum() តាមរយៈ obj2 នៃ sample class

**}**

ឧទាហរណ៍ទី ២ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត និងប្រកាស class template ដោយមាន default constructor។

**// using class template**

**#include <iostream.h>**

**template<class T>**

**class sample {**

**private :**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T មានលក្ខណៈ private

**public :**

**sample(T = 0) { } // default constructor**

**void display(){**

**cout << "Default constructor is called." << endl;**

**cout << "Contents of the value = " <<value <<endl;**

**}**

**};**

**void main(){**

**sample <int> obj1; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

**obj1.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj1 នៃ sample class

**sample <float> obj2; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj2

**obj2.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj2 នៃ sample class

**}**

ឧទាហរណ៍ទី ៣ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត និងប្រកាស class template ដោយមាន constructor member function។

**// using class template defining constructor**

**#include <iostream.h>**

**template<class T>**

**class sample {**

**private :**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T មានលក្ខណៈ private

**public :**

**sample() {} // constructor**

**void display(){**

**cout << "Constructor is called." << endl;**

**cout <<"Contents of the value = "<<value << endl;**

**}**

**};**

**void main(){**

**sample <int> obj1; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

**obj1.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj1 នៃ sample class

**sample <float> obj2; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj2

**obj2.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj2 នៃ sample class

**}**

ឧទាហរណ៍ទី ៤ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត និងប្រកាស class template ដោយមាន member function, constructor និង destructor ពិសេស។

**// using class template defining constructor and destructor**

**#include <iostream.h>**

**template<class T>**

**class sample {**

**private :**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T មានលក្ខណៈ private

**public :**

**sample(){} //** ការប្រកាស constructor

**~sample(){} //** ការប្រកាស destructor

**void display(){**

**cout << "Constructor is called." << endl;**

**cout <<"Contents of the value = "<<value << endl;**

**}**

**};**

**void main(){**

**sample <int> obj1; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

**obj1.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj1 នៃ sample class

**sample <float> obj2; //** ការបង្កើត object នៃ sample class ឈ្មោះ obj2

**obj2.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj2 នៃ sample class

**}**

ឧទាហរណ៍ទី ៥ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត និងប្រកាស class template ដោយមាន member function, constructor និង destructor ពិសេស។ ហើយ constructor មាន argument រួមមួយ។

**// constructor with single argument**

**// and destructor of class template**

**#include <iostream.h>**

**template<class T>**

**class sample {**

**private :**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T មានលក្ខណៈ private

**public :**

**sample (T n):value(n) {}; //** ការប្រកាស constructor

**~sample(){ } //** ការប្រកាស destructor

**void display(){**

**cout <<"Contents of the value = "<< value<<endl;**

**}**

**};**

**void main(){**

**sample <int> obj1(10); //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

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

**obj1.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj1 នៃ sample class

**sample <float> obj2(-22.12345);**

**//** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

**cout << "Floating point number :" << endl;**

**obj2.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj2 នៃ sample class

**}**

ឧទាហរណ៍ទី ៦ ៖ កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបបង្កើត និងប្រកាស class template ដោយមាន member function, constructor និង destructor ពិសេស។ ហើយ constructor មាន argument រួមមួយដោយមានទម្រង់ផ្សេងគ្នា។

**// constructor with single argument**

**// and destructor of class template**

**#include <iostream.h>**

**template <class T>**

**class sample {**

**private :**

**T value; //** ការប្រកាសអញ្ញាត value ជាប្រភេទ T មានលក្ខណៈ private

**public :**

**sample(T n); //** ការប្រកាស constructor

**~sample(); //** ការប្រកាស destructor

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

**};**

**template <class T>**

**sample <T>::sample(T n):value(n) {}**

**template <class T>**

**sample <T>::~sample(){}**

**template <class T>**

**void sample<T>::display(){**

**cout << "Contents of the value = " << value << endl;**

**}**

**void main(){**

**sample <int> obj1(10); //** ការបង្កើត object នៃ sample class ឈ្មោះ obj1

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

**obj1.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj1 នៃ sample class

**sample <float> obj2(-22.12345);**

**//** ការបង្កើត object នៃ sample class ឈ្មោះ obj2

**cout << "Floating point number : " << endl;**

**obj2.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj2 នៃ sample class

**sample <double> obj3(12345678L);**

**//** ការបង្កើត object នៃ sample class ឈ្មោះ obj3

**cout << "Double precision number :" << endl;**

**obj3.display(); //** ការហៅអនុគមន៍ display() តាមរយៈ obj3 នៃ sample class

**}**

៣. ការប្រើ Exception

exception គឺជា error មួយឬអំពើមួយដែលកើតឡើងដោយមិនអាចគិតជាមុនបាន។ ការប្រើ exception គឺជាសំណុំ code ដែលប្រតិបត្តិនៅពេល exception មួយកើតឡើង។ វាជាលក្ខណៈពិសេសថ្មីដែលទើបបញ្ចូល ហើយប្រហែលជាវាមិនអាចប្រើបានចំពោះ version មុនៗ​ នៃ C++ compilers ឡើយ។

ការប្រើ exception នៅក្នុង C++ ផ្តល់វិធីមួយដ៏ប្រសើរដោយហៅអនុគមន៍មកប្រើអាចឲ្យព័ត៌មានពីលក្ខខណ្ឌ error មួយចំនួនដែលនឹងកើតឡើង។ ពាក្យសំខាន់ៗដែលត្រូវបានប្រើសម្រាប់អនុវត្ត error functions នៅក្នុង C++ មានដូចខាងក្រោម។

try

catch

throw

ទោះជាការហៅ អនុគមន៍មកប្រើរកឃើញ error មួយក៏ដោយ វាមានការពិបាកផ្ទៀងផ្ទាត់ជាមួយនឹងសញ្ញា error ទាំងនេះណាស់។ នៅក្នុង C++, លក្ខខណ្ឌ error ប្រភេទទាំងនេះអាចអនុវត្តបានយ៉ាងងាយ ដោយប្រើពាក្យសំខាន់ៗខាងលើនេះ។

នៅពេលការហៅអនុគមន៍មកប្រើបានប្រទះឃើញ error មួយដោយមិនបានប្រើ exception នោះ វាមានការពិបាកប្រើនៅក្នុង software ធំៗនិងសាំញ៉ាំ។ កម្មវិធីត្រូវតែបង្កើតឡើងដោយប្រើ exception តាមវិធីបែបនេះធ្វើឲ្យវាកំណត់បាននូវ error ដែលកម្មវិធីអាចជួបប្រទះ រួចហើយបញ្ចូល code ដើម្បីប្រើវា។ ឧទាហរណ៍នៅក្នុងកម្មវិធីដែលធ្វើការប្រតិបត្តិ input / output នៅក្នុងដំណើរការនៃ file នោះ វាត្រូវការយ៉ាងខ្លាំងដើម្បីពិនិត្យថាតើ file មួយបើកបានជោគជ័យឬមិនជោគជ័យ ហើយបង្ហាញឃ្លា error បានត្រឹមត្រូវ បើសិនវាបានកើតឡើង error នោះ។

ការប្រើ exception ផ្តល់នូវវិធីមួយទៀត ដើម្បីផ្ទេរការត្រួតពិនិត្យ និងព័ត៌មានពីចំណុចមួយនៅក្នុងការប្រតិបត្តិនៃកម្មវិធីទៅជាការប្រើ exception។ ការអនុវត្តនឹងត្រូវបានយកមកប្រើតាមរយៈឃ្លា throw នៅក្នុង code ដែលបានប្រតិបត្តិនៅក្នុង try block ឬ អនុគមន៍ដែលបានហៅពី try block។

ទម្រង់ទូទៅនៃ try និង catch មានដូចខាងក្រោម ៖

try {

*// try block*

}

catch *(type1 arg)* {

*// catch block*

}

catch *(type2 arg)* {

*// catch block*

}

*// ...*

catch *(typeN arg)* {

*// catch block*

}

ទម្រង់ទូទៅនៃឃ្លា throw មានដូចខាងក្រោម ៖

throw *exception;*

ឧទាហរណ៍ code ខាងក្រោមនេះបង្ហាញពីរបៀបបង្កើតនិងប្រកាសប្រើ exception ក្នុងកម្មវិធីមួយ។ ការប្រើ exception គឺជា error ប្រភេទ run time ដែលបណ្តាលមកពីលក្ខខណ្ឌមិនប្រក្រតី។

**class sample {**

**private :**

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

**public :**

**enum {minsize = 1, maxsize = 1000};**

**sample(); //** ការប្រកាស constructor គ្មានប៉ារ៉ាម៉ែត្រ

**sample(int); //** ការប្រកាស constructor មានមួយប៉ារ៉ាម៉ែត្រ

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

**};**

**sample::sample(int size){**

**if (size < minsize || size > maxsize)**

**throw (size);**

**str = new char[size];**

**if (str == 0) //** បើសិន str មានតម្លៃស្មើសូន្យនោះ

**throw ("Out of memory\n");**

**}**

**void funct(int n){**

**try {**

**sample obj(n); //** ការបង្កើត object នៃ sample class ឈ្មោះ obj

**} catch (int k) {**

**cerr << "Out of range ... \n";**

**funct(sample::maxsize);**

**}**

**} // end of funct definition part**

កម្មវិធីខាងក្រោមនេះជាឧទាហរណ៍មួយទៀតដែលបង្ហាញពីការប្រើ exception ដ៏សាមញ្ញបំផុត។

**// A simple exception handling example.**

**#include <iostream.h>**

**void main(){**

**cout << "start\n"; //** បង្ហាញអក្សរ start មកលើអេក្រង់

**try { // start a try block**

**cout << "Inside try block.\n";**

**//** បង្ហាញអក្សរ Inside try block.មកលើអេក្រង់

**throw 99; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int**

**cout << "This will not execute.";**

**}**

**catch (int i) { // វានឹងចាប់យក** error **ប្រភេទ int**

**cout << "Caught an exception -- value is: ";**

**cout << i << "\n"; //** បង្ហាញតម្លៃរបស់អញ្ញាត i មកលើអេក្រង់

**}**

**cout << "end";**

**}**

កម្មវិធីខាងក្រោមនេះមិនអាចចាប់យក error បានឡើយ ដោយសារវាមានប្រភេទទិន្នន័យខុសគ្នា រវាង throw និង catch។

**// This example will not work.**

**#include <iostream.h>**

**void main(){**

**cout << "start\n"; //** បង្ហាញអក្សរ start មកលើអេក្រង់

**try { // start a try block**

**cout <<"Inside try block\n";//** បង្ហាញអក្សរ Inside try block មកលើអេក្រង់

**throw 99; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int**

**cout << "This will not execute";**

**}**

**catch (double i) { // វាមិនដំណើរការចំពោះ exception ចាប់យកប្រភេទ** int **ឡើយ**

**cout << "Caught an exception -- value is: ";**

**cout << i << "\n"; //** បង្ហាញតម្លៃរបស់អញ្ញាត i មកលើអេក្រង់

**}**

**cout << "end";**

**}**

កម្មវិធីខាងក្រោមនេះបង្ហាញពីការប្រើ exception ដែលបាន throw តាមរយៈអនុគមន៍មួយ ត្រូវបានហៅប្រើនៅក្នុង *try* block ហើយអាចចាប់យក exception នោះដោយ *try* block នោះ។

**/\* Throwing an exception from a function called**

**from within a try block. \*/**

**#include <iostream.h>**

**void Xtest(int test){**

**cout << "Inside Xtest, test is: " << test << "\n";**

**if(test) throw test;**

**}**

**void main(){**

**cout << "start\n"; //** បង្ហាញអក្សរ start មកលើអេក្រង់

**try{ // start a try block**

**cout << "Inside try block\n";**

**Xtest(0); //** ការហៅអនុគមន៍ Xtest() មកប្រើ

**Xtest(1); //** ការហៅអនុគមន៍ Xtest() មកប្រើ

**Xtest(2); //** ការហៅអនុគមន៍ Xtest() មកប្រើ

**} catch(int i){ // វានឹងចាប់យក** error **ប្រភេទ int**

**cout << "Caught an exception -- value is: ";**

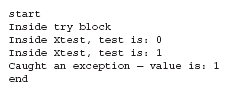
**cout << i << "\n"; //** បង្ហាញតម្លៃរបស់អញ្ញាត i មកលើអេក្រង់

**}**

**cout << "end";**

**}**

លទ្ធផលរបស់វាគឺ ៖



try block អាចប្រើនៅក្នុងអនុគមន៍។ កាលណាគេធ្វើបែបនេះ ការប្រើ exception ដែលទាក់ទងទៅនឹងអនុគមន៍នោះត្រូវកំណត់សាជាថ្មីរៀងរាល់ពេលចូលដល់អនុគមន៍។ ចូរពិនិត្យឧទាហរណ៍ខាងក្រោមនេះ។

**#include <iostream.h>**

**// A try/catch is reset each time a function is entered.**

**void Xhandler(int test){**

**try{**

**if(test) // បើសិនជា test ពិតនោះ**

**throw test; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int**

**} catch(int i) { // វានឹងចាប់យក** error **ប្រភេទ int**

**cout << "Caught One! Ex. #: " << i << '\n';**

**}**

**}**

**void main(){**

**cout << "start\n";**

**Xhandler(1); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(2); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

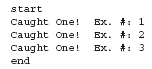
**Xhandler(0); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(3); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**cout << "end";**

**}**

លទ្ធផលនៃកម្មវិធីខាងលើនេះគឺ ៖



៣.១ ការប្រើ Class Types សម្រាប់ចាប់ exception

Exception អាចជាប្រភេទទិន្នន័យណាមួយក៏បាន រួមទាំង class types ដែលយើងបង្កើតឡើងនោះផងដែរ។ ឧទាហរណ៍ខាងក្រោមនេះបង្ហាញពីលក្ខណៈនេះ។

​ **// Use an exception class.**

**#include <iostream.h>**

**#include <string.h>**

**class MyException {**

**public:**

**char str\_what[80];**

**//** ការប្រកាសអញ្ញាត str\_what ជា array ប្រភេទ char មានទំហំ 80 តួអក្សរ

**MyException(){ //** ការប្រកាស constructor ដោយកំណត់តម្លៃសូន្យទៅឲ្យ str\_what

**\*str\_what = 0;**

**}**

**MyException(char \*s){**

**strcpy(str\_what, s); //** ការប្រើអនុគមន៍ strcpy

**}**

**};**

**void main(){**

**int a, b; //** ការប្រកាសអញ្ញាត a, b ជាប្រភេទ int

**try {**

**cout << "Enter numerator and denominator: ";**

**cin >> a >> b; //** បញ្ចូលតម្លៃ a, b តាមរយៈ keyboard

**if (b==0) // បើសិនជា b មានតម្លៃស្មើសូន្យ នោះ**

**throw MyException("Cannot divide by zero!");**

**else**

**cout << "Quotient is " << a/b << "\n";**

**}**

**catch (MyException e){ // catch an error**

**cout << e.str\_what << "\n";**

**}**

**}**

ការប្រើឃ្លា catch ច្រើន ៖

**// A try/catch is reset each time a function is entered.**

**void Xhandler(int test){**

**try {**

**if(test) // បើសិនជា test ពិតនោះ**

**throw test; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int**

**} catch(int i){ // វានឹងចាប់យក** error **ប្រភេទ int**

**cout << "Caught One! Ex. #: " << i << '\n';**

**}**

**}**

**void main(){**

**cout << "start\n";**

**Xhandler(1); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(2); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(0); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(3); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**cout << "end";**

**}**

ការចាប់យក exceptions របស់ Base Class ៖ វាមានចំណុចសំខាន់មួយអំពីឃ្លា catch ច្រើនដែលទាក់ទងទៅនឹង derived classes។ ឃ្លា catch សម្រាប់ base class នឹងត្រូវគ្នាជាមួយ derived class ណាមួយរបស់ base class បានដែរ។ ហេតុនេះ បើសិនជាគេចង់ចាប់ exceptions របស់ប្រភេទ base class និង derived class នោះត្រូវដាក់ derived class មុនគេនៅក្នុងលំដាប់ catch។ បើមិនដូច្នោះទេ ឃ្លារបស់ base class នឹងចាប់យកប្រភេទ derived class ទាំងអស់។ ចូរសង្កេតឧទាហរណ៍ខាងក្រោមនេះ ៖

**// Catching derived classes.**

**#include <iostream.h>**

**class B {**

**};**

**class D: public B {**

**};**

**void main(){**

**D derived;**

**try {**

**throw derived; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ D class**

**} catch(B b) { // វានឹងចាប់យក** error **ប្រភេទ B class**

**cout << "Caught a base class.\n";**

**} catch(D d) { // វានឹងចាប់យក** error **ប្រភេទ D class**

**cout << "This won't execute.\n";**

**}**

**}**

៣.២ ការចាប់យកគ្រប់ exceptions ទាំងអស់

ទម្រង់ទូទៅរបស់វា គឺ ៖

catch(...) {

// *process all exceptions*

}

ដែលក្នុងនេះ សញ្ញាចំណុចត្រូវគ្នានឹងគ្រប់ប្រភេទទិន្នន័យ។ កម្មវិធីខាងក្រោមនេះបង្ហាញពីការប្រើវា។

**// This example catches all exceptions.**

**#include <iostream.h>**

**void Xhandler(int test){**

**try{**

**if(test==0) throw test;**

**// វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int កាលណា test ស្មើនឹងសូន្យ**

**if(test==1) throw 'a';**

**// វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ char កាលណា test ស្មើនឹងមួយ**

**if(test==2) throw 123.23;**

**// វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ double កាលណា test ស្មើនឹងពីរ**

**} catch(...) { // វានឹងចាប់យក** error **គ្រប់ប្រភេទ**

**cout << "Caught One!\n";**

**}**

**}**

**void main(){**

**cout << "start\n";**

**Xhandler(0); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(1); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(2); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**cout << "end";**

**}**

កម្មវិធីខាងក្រោមនេះជារបៀបមួយទៀតដែលមានការប្រើ catch(...) នៅផ្នែកខាងចុងនៃការប្រើ catch មួយទៀត ដែលជាលក្ខណៈមួយល្អប្រសើរ។

**// This example uses catch(...) as a default.**

**#include <iostream.h>**

**void Xhandler(int test){**

**try{**

**if(test==0)**

**throw test; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int**

**if(test==1)**

**throw 'a'; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ char**

**if(test==2)**

**throw 123.23; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ double**

**} catch(int i) { // វានឹងចាប់យក** error **ប្រភេទ int**

**cout << "Caught " << i << '\n';**

**} catch(...) { // វានឹងចាប់យក** error **គ្រប់ប្រភេទ**

**cout << "Caught One!\n";**

**}**

**}**

**void main(){**

**cout << "start\n";**

**Xhandler(0); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(1); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**Xhandler(2); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**cout << "end";**

**}**

៣.៣ ការកម្រិត exceptions ដែលត្រូវ throw ដោយ function មួយ

គេអាចកម្រិតប្រភេទ exceptions ដែលអនុគមន៍មួយអាច throw ខាងក្រៅខ្លួនវាបាន។ ទម្រង់ទូទៅរបស់វា គឺ ៖

*ret-type func-name*(*arg-list*) throw(*type-list*) {

// ...

}

កម្មវិធីខាងក្រោមនេះបង្ហាញនូវរបៀបកម្រិតប្រភេទ exceptions ដែលអាច thrown ពី អនុគមន៍មួយ។

**// Restricting function throw types.**

**#include <iostream.h>**

**// This function can only throw ints, chars, and doubles.**

**void Xhandler(int test) throw(int, char, double){**

**if(test==0)**

**throw test; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ int**

**if(test==1)**

**throw 'a'; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ char**

**if(test==2)**

**throw 123.23; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ double**

**}**

**void main() {**

**cout << "start\n";**

**try {**

**Xhandler(0); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**} catch(int i) { // វានឹងចាប់យក** error **ប្រភេទ int**

**cout << "Caught int\n";**

**} catch(char c) { // វានឹងចាប់យក** error **ប្រភេទ char**

**cout << "Caught char\n";**

**} catch(double d) { // វានឹងចាប់យក** error **ប្រភេទ double**

**cout << "Caught double\n";**

**}**

**cout << "end";**

**}**

ការ throw exception សាជាថ្មី

**// Example of "rethrowing" an exception.**

**#include <iostream.h>**

**void Xhandler(){**

**try {**

**throw "hello"; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ char \***

**} catch(char \*) { // វានឹងចាប់យក** error **ប្រភេទ char \***

**cout << "Caught char \* inside Xhandler.\n";**

**throw ; // វានឹងធ្វើឲ្យមាន** error **មួយកើតឡើងជាប្រភេទ char \* ជាថ្មីម្តងទៀត**

**}**

**}**

**void main(){**

**cout << "start:\n"; // បង្ហាញអក្សរ start មកលើអេក្រង់**

**try {**

**Xhandler(); //** ការហៅអនុគមន៍ Xhandler() មកប្រើ

**} catch(char \*) { // វានឹងចាប់យក** error **ប្រភេទ char \***

**cout << "Caught char \* inside main\n";**

**}**

**cout << "end";**

**}**

សំណួរនិងលំហាត់

១ - ចូរនិយាយពីលក្ខណៈ template និងសារៈសំខាន់របស់វា។

២ - តើ template និង overloaded functions ខុសគ្នាដូចម្តេច?

៣ - ចូរសរសេរ code បង្ហាញពីការប្រើ function template មួយសម្រាប់រៀបតម្លៃតាមលំដាប់

កើនឡើង។

៤ - ចូរសរសេរ code បង្ហាញពីការប្រើ class template មួយ។

៥ - អ្វីទៅ Exception? ចូរនិយាយពីមុខងារ try, catch, និង throw ។

៦ - ចូរសរសេរ code បង្ហាញពីការប្រើ Exception សម្រាប់ចាប់យក error គឺការចែកចំនួនមួយទៅនឹងសូន្យ។

៧ - ចូរសរសេរ code បង្ហាញពីការប្រើ Exception សម្រាប់ចាប់យក error កើតឡើងគ្រប់ប្រភេទ។

៨ - ចូរបង្កើត function template មួយដែលអាចប្រើបាន ដូច overloaded functions ខាងក្រោម ៖

|  |  |
| --- | --- |
| void maxmin(int a[ ], int n){  int min, max;  min = max = a[0];  for (int i=1; i<n; i++){  if (min > a[i])  min = a[i];  if (max < a[i])  max = a[i];  }  cout << "min = " << min << endl;  cout << "max = " << max << endl;  } | void maxmin(float a[ ], int n){  float min, max;  min = max = a[0];  for (int i=1; i<n; i++){  if (min > a[i])  min = a[i];  if (max < a[i])  max = a[i];  }  cout << "min = " << min << endl;  cout << "max = " << max << endl;  } |