អំពីអនុគមន៏ (Functions)​

១. សេចក្តីផ្តើម

បញ្ហាសាំញ៉ាំមួយត្រូវបានបំបែកទៅជា ​modules ឬជាចំណែកតូចៗដើម្បីងាយស្រួល គ្រប់គ្រង់ហៅថា function ។ functions មានសារ:សំខាន់ដល់ការអាន សរសេរ កែតម្រូវកម្មវិធី សាំញ៉ាំ។ វាអាចធ្វើការបានយ៉ាងងាយជាមួយកម្មវិធីចំបង (main program) ។ នៅក្នុង c++ main() ខ្លួនវាក៏ជា function មួយដែល ហើយបង្កអោយ function ដ៏ទៃទៀតដំណើរការងារ​ ​ ផ្សេងៗគ្នាបាន។

សារ:សំខាន់នៃការប្រើ function គឺ

.​​ ផ្តល់លក្ខណ:ងាយស្រួលដល់ការសរសេរ function តូចៗបានត្រឹមតរូវ

. ផ្តល់ភាពងាយស្រួលក្នុងការអាន សរសេរ និងកែតំរូវ function

. ផ្តល់ភាពងាយស្រួលដល់ការទុកដាក់ និងកែប្រែ function

. វាអាចហៅប្រើពេលណាក៏បាន ទោះជាទៅទីណាក៏ដោយជាមួយ នឹងប៉ារ៉ាម៉ែត្រផ្សេងៗគ្នា។

២. និយមន័យអនុគមន៏ (function)

អនុគមន៏ គឺជាសំណុំនៃឃ្លាដែលត្រូវធ្វើការប្រតិបត្តិលើការងារមួយនៅពេលហៅវាមក ប្រើតាមរយ:ឈ្មោះរបស់វានៅត្រង់ចំនុចណាមួយនៃកម្មវិធី។ វាមានទម្រង់ទូទៅដូចខាងក្រោម

functiontype functionName(datatype argu1, datatype argu2, … ) {

statements

…………….

Return something

}

ដែលក្នុងនេះ functiontype គឺជាប្រភេទទិន្នន័យត្រូវគ្នានឹងតម្លៃដែលអោយដោយ function នោះ។ functionName គឺជាឈ្មោះប្រើសម្រាប់ហៅ function នោះ។ arguments គឺអាចមានចំនួនមិតកំណត់។​​ argument និមួយៗមានប្រភេទទិន្នន័យដោយខ្លួនដូចទៅនឹងការប្រកាស អញ្ញាតដែរ។ ហើយវាធ្វើសកម្មភាពនៅក្នុង function ដូចអញ្ញាតផ្សេងទៀត។ វាអាចអោយគេ បញ្ចូលតម្លៃ ប៉ារ៉ាម៉ែត្រទៅកាន់ function នៅពេល function នោះត្រូវបានហៅមកប្រើ។ ប៉ារ៉ាម៉ែត្រផ្សេងគ្នាត្រូវញែកដាច់ពីគ្នាដោយសញ្ញាក្បៀស។ statement គឺជាតួនៃ function ។ វាអាចជា instruction មួយ ឬច្រើន។ something គឺជាតម្លៃមួយ ឬជាកន្សោមមួយដែល function នឹង​ ផ្តល់អោយបន្ទាប់ពីដំណើរការបញ្ចប់។

ឧទាហរណ៏​ទី១

#include<iostream.h>

int addition(int a , int b) {

int r;

r= a + b;

return (r) ;

}

void main() {

int x ;

x = addition ( 11 , 22 );

cout << “ The result is = “<< x << endl;

}

ជាបឋម យើងគួរយល់នូវ code ខាងលើនេះនិងចង់ចាំថាកម្មវិធី C++ ជានិច្ចកាលចាប់ ផ្តើមប្រតិបត្តិការជាមួយ main() function ។ ដូចនេះយើងត្រូវពិនិត្យត្រង់ចំណុចនេះទៅ។ នៅក្នុង main() គេឃើញមានការប្រកាសអញ្ញាត x ជាប្រភេទ int ។ បន្ទាប់មកមានការហៅមកប្រើ addition function ។ យើងគូរយកចិត្តទុកដាក់ត្រង់ addition function នេះមានទំរង់ដូចគ្នាទៅនឹង ទំរង់ function ដែលបានប្រកាសខាងលើវា។

Int addition ( int a , int b )

x = addition ( 11 , 22 );

នៅត្រង់ចំនុចនេះវាបញ្ចូលពីរតម្លៃគឺ 11 និង​ 22 តាមរយ:ប៉ារ៉ាម៉ែត្រ a និង​ b ។ តម្លៃនេះ ត្រូវបានចម្លងទៅអោយអញ្ញាតខាងក្នុង (Local variable) ។ រីឯ addition() function វិញមានការ ប្រកាសអញ្ញាតថ្មីមួយគឺ x ជាប្រភេទ int ហើយត្រូវបានកំណត់តម្លៃទៅអោយកន្សោម x=a+b ។ ដោយ a ទទួលបានតម្លៃ 11 និង​ b ទទួលបានតម្លៃ 22 នោះវាអោយលទ្ធផល 33 ផ្ទុកក្នុង x។ តម្លៃ 33 នេះត្រូវបានបញ្ចូនទៅអោយ x ក្នុង main() តាមការកំណត់តម្លៃ x = addition (11,22);

int addition ( int a , int b)

x = addition ( 11 , 22);

ឧទាហរណ៏ទី២

#include<iostream.h>

void stars(void);

void main() {

char text[20]=” some text “;

stars();

cout << endl;

cout << ‘ \* ‘ ;

cout << text;

cout << ‘ \* ‘ << endl;

stars();

cout << endl;

}

void stars () {

for ( int i =0; i < = 10 ; i++ )

cout << ‘ \* ‘ ;

}

២.១ ការប្រកាសប្រភេទទិន្នន័យរបស់ function

Function បញ្ជាក់អោយប្រភេទតម្លៃដែលវាផ្តល់អោយទៅកាន់ផ្នែកណាមួយនៃកម្មវិធី ដែលប្រើវា។ ប្រភេទទិន្នន័យធម្មតា (ដូចជា int , float , char ជាដើម) អាចកើតមានក្នុងពេល ប្រកាស function ។ កាលណា function មួយមិនមានការអោយតម្លៃទេនោះគេអាចប្រកាសជា ប្រភេទ void ដែលបញ្ជាក់ដល់ conpiler អោយដឹងថាមិនបាច់រក្សាទុកទីតាំងផ្ទុកតម្លៃដែលត្រូវ បញ្ចូនទៅអោយកម្មវិធីដែលប្រើវានោះ។

ឧទាហរណ៏

void function\_name (…)

int function\_name (…)

float function\_name (…)

char function\_name (…)

២.២ ឈ្មោះ function

ឈ្មោះ function អាចជាឈ្មោះណាមួយដោយធានាអះអាងទៅតាមក្បួនបែបបទរបស់ អញ្ញាត។ តាមធម្មតាឈ្មោះ function មួយបង្កើតឡើងទាក់ទងទៅនឹងការប្រតិបត្តិរបស់ function ដើម្បីងាយស្រួលកំណត់ចំណាំវា។

ឧទាហរណ៏

factorial ()

intput ()

output()

display\_value ()

២.៣ តួនៃ function ( function body)

បន្ទាប់ពីប្រកាសប្រភេទទិន្នន័យនៃ function ឈ្មោះរបស់ function និង ​formal arguments ពេលនោះ statement មួយ ឬឃ្លាច្រើនត្រូវដាក់ក្នុងចន្លោះសញ្ញាចាប់ផ្តើម “{” និង សញ្ញាបញ្ចប់ “}” ។

ឧទាហរណ៏ទី១

#include<iostream.h>

void display ( void );

void main() {

display();

}

void display() {

cout << “ This is a test program \n”;

}

ឧទារហណ៏ទី២

#include<iostream.h>

void sum (int a , in t b);

void main() {

int x ,y;

cout << “Enter x = “; cin >> x;

cout << “Enter y = “; cin >> y;

sum(x,y);

}

void sum( int a, int b) {

int s;

s= a + b;

cout << “ Sum = “ << s << endl;

}

៣.​ការប្រើឃ្លា return

ពាក្យ return ត្រូវបានប្រើសម្រាប់បញ្ចប់ និងអោយតម្លៃមួយទៅកាន់អ្នកហៅវាមកប្រើ ឃ្លា return នេះក៏អាចប្រើសម្រាប់ចាកចេញពី function មួយដោយមិនអោយតម្លៃបានដែរ។ ឃ្លា return អាចប្រើជាមួយកន្សោម ឬមិនប្រើជាមួយកន្សោម។ ទម្រង់ទូទៅរបស់វាគឺ

return ;

return(expression);

ពាក្យ return អាចប្រើនៅកន្លែងណាក៏បានក្នុងតួរបស់ function ។ function មួយអាចប្រើ return មួយ ឬច្រើនទោះបីជាវាមានលំនាំសរសេរកម្មវិធីល្អចំពោះ function មួយដែលមានតម្លៃ បញ្ចូល និងការចាកចេញតែមួយគត់ក៏ដោយ។

ឧទាហរណ៏

return ;

return (71);

return ( x + y);

return ( ++ i);

ឃ្លា return ខាងក្រោមនេះបញ្ចប់ការប្រតិបត្តិនៃ function ហើយដំណើរវិលទៅរកមជ្ឈដ្ឋានដែលគេហៅប្រើនោះ។

ឧទាហរណ៏ទី១

int sum ( int x , int y) {

return (x+y);

}

ឧទាហរណ៏ទី២

#include<iostream.h>

int minimum ( int a , in b );

void main() {

int val1, val2 , min;

cout << “Enter val1 = “ ; cin >> val1;

cout << “Enter val2 = “ ; cin >> val2;

min = minimum ( val1 , val2 );

cout << “ Minimum of “ << val1 << “ and “ << val2

<< “ is “ << min << endl;

}

int minimum ( int a , int b ) {

if ( a < b )

return a;

else

return b;

}

ឧទាហរណ៏ទី៣

#include< iostream.h>

void main () {

float maximum (float ,float , float );

float x, y , z, max;

cout << “Enter three numbers \n” ;

cin >> x >> y >> z ;

max = maximum ( x, y ,z );

cout << “ Maximum = “ << max;

}

1. float maximum ( float a, float b, float c ) {

if ( a > b ) {

if (a > c)

return (a);

else

return (c);

}

else {

if (b > c)

return (b);

else return (c);

}

}

1. float maximum (float a , float b, float c) {

float m;

m = a;

if (m<b)

m = b;

if (m < c)

m = c;

return m;

}

៤. ប្រភេទ functions

Function ដែលបង្កើតឡើងដោយអ្នកប្រើប្រាស់ត្រូវបានចំណាត់ថ្នាក់តាមបីវិធីដោយ ផ្អែកលើការបញ្ជូនតម្លៃរបស់ formal arguments និងការប្រើឃ្លា return ។

**ប្រភេទទី១:** function មួយបង្កឡើងដោយគ្មានការបញ្ជូនតម្លៃ formal argument ណា សោះ តាមរយ:ផ្នែកណាមួយនៃកម្មវិធី ហើយ function ពុំមានការអោយតម្លៃក្នុងពេលហៅវាមក ប្រើ​ ។

ឧទាហរណ៏

#include< iostream.h>

void myFunction (void);

void main () {

int i;

for ( i = 0 ; i < 5; i++)

myFunction ();

}

void myFunction () {

static int var =10;

cout << var << endl;

var + = 2;

}

លទ្ធផលកម្មវិធីខាងគឺ 10 12 14 16 18

**ប្រភេទទី២:** function មួយបង្កឡើងដោយមានការបញ្ជូនតម្លៃតាម formal arguments នៅពេលដែលផ្នែកណាមួយនៃកម្មវិធីបានហៅមកប្រើ ក៏ប៉ុន្តែ function ពុំមានការអោយតម្លៃទៅអ្នកហៅវាមកប្រើឡើយ​ ។

//passing formal arguments and no return statement

#include<iostream.h>

void square ( int n) {

float value;

value = n \* n ;

cout << “ n “ << n << “ square = “<< value << endl;

}

void main () {

int max;

cout << “ Enter a value for n \n” ;

cin >> max;

for ( int i=0; i < max ; i++)

square(i);

}

ប្រភេទទី៣: function មួយបង្កឡើងដោយមានការបញ្ជួនតម្លៃតាម​ formal arguments នៅពេលដែលផ្នែកណាមួយនៃកម្មវិធីហៅវាមកប្រើ ហើយវានឹងអោយតម្លៃមួយទៅកាន់ អ្នកហៅប្រើនោះ។

ឧទាហរណ៏ទី១:

// using return statement

#include<iostream.h>

float square (float n ) {

float value;

value = n \* n;

return (value);

}

void main () {

float i = -2.5, max = 2.5, value;

while ( i < = max ) {

value = square(i);

cout << “ i = “ << i << “ square = “ << value <<endl;

i = i + 0.5;

}

}

ឧទាហរណ៏ទី២:

// factorial of a given number

#include<iostream.h>

long int fact ( int n ) {

int value =1;

if ( n == 1)

return (value);

else {

for (int i = 1; i < =n ; i++)

value = value \* i;

return (value);

}

}

void main () {

long int x;

int n;

cout << “ Enter any integer number : “ << endl;

cin >> n;

x = fact (n);

cout << “value = “ << n << “ and its factorial = “ << x << endl;

}

ឧទាហរណ៏ទី៣:

// sum = 1 + 2 + 3 + …… + n using non-recursiv function

#include<iostream.h>

int sum(int n ) {

int value =0;

if ( n == 0 )

return (value);

else {

for (int i=1; i <= n ; i++)

value +=i;

return (value);

}

}

void main() {

int n;

cout << “ Enter any integer number : “ <<endl;

cin >> n ;

cout << “ value “ << n << “ and its sum = “ << sum (n) ;

}

៥.​ អ្វីទៅហៅថា formal arguments ?

Formal argument គឺជាប៉ារ៉ាម៉ែត្រដែលគេសរសេរដាក់នៅ function header ក្នុងពេល ប្រកាស ហើយគេហៅវាម៉្យាងទៀតថា dummy arguments ឬ parametric variables ។

ឧទាហរណ៏ទី១

void square ( int a, int b ) { // a , b are the formal arguments

…………

…………

}

ឧទាហរណ៏ទី២

Int counter (float x1, float x2, int y1, int y2 ) {

// x1, x2 , y1, and y2 are the formal arguments

…………

…………

return (int value );

}

ឧទាហរណ៏ទី៣

float output () {

statement ;

……………

……………

return (float value );

}

ឧទាហរណ៏ទី៤

// void function example without formal argument

#include<iostream.h>

void dummyfunction() {

cout << “ I’m a function ! “ ;

}

int main () {

Dummyfunction ();

return 0;

}

ឧទាហរណ៏ទី៥

// function arguments

#include<iostream.h>

void main() {

char text[29] = “ Royal University of Phno Penh” ;

stars (5);

cout << endl;

stars (1);

cout << endl;

stars ( 5);

cout << endl;

}

Void stars (int n) {

for ( int counter = 0 ; counter < n ; counter++)

cout << ‘ \* ‘ ;

}

ឧទាហរណ៏ទី៦

#include<iostream.h>

int subtract ( int a , int b ) {

int r;

r = a – b ;

return (r);

}

void main() {

int x = 5, y = 3 , z ;

z = subtract ( 7 , 2 );

cout << “ The first result is “ << z << endl;

cout << “ The Second result is “ << subtract( 7 , 2 ) <<endl;

cout << “ The third result is “ << subtract ( x, y ) << endl;

z = 5 + subtract (x , y );

cout << “ The fourth result is “ << z << endl;

}

**៦.​ អំពី actual arguments**

Actual argument គឺជាអញ្ញាតមួយ ឬកន្សោមមួយដែលបានដាក់ក្នុង function ពេលហៅមកប្រើ ហើយវាជំនួស formal parameter ដែលជាផ្នែកមួយនៃការប្រកាស​​ function​។

ឧទាហរណ៏

#include<iostream.h>

void output ( int a , int b ) {

…………….

}

void main () {

Int x , y ;

………

………

Output( x, y );

}

៧. ការបញ្ជូន​ arguments តាមលក្ខណ: by value (Passing argument ny value)

Function ដែលប្រើលក្ចណ:ចម្លងតម្លៃរបស់ argument ដើម្បីបញ្ជូនតាមប៉ារ៉ាម៉ែត្ររបស់ function ហៅថាការបញ្ជូន arguments ដោយតម្លៃ​ ។ តម្លៃនៅក្នុង argument មិនមានការប្រែប្រួលឡើយក្រោយពីវាបញ្ជូនចូលទៅក្នុង function នោះ​ ។

ឧទាហរណ៏

// passing arguments by value.

#include<iostream.h>

void any\_function (int p) ;

void main() {

int a = 1;

cout << “ a is “ << a << endl;

any\_function(a);

cout << “ a is still “ << a << endl;

}

Void any\_function ( int p ) {

cout << “ p is “ << p << endl;

p = 3;

}

៨. ការបញ្ជួន arguments តាមលក្ខណ: by reference (Passing arguments by reference)

Reference មានអត្ថន័យប្រៀបបានដូចអញ្ញាតមួយដែលមានឈ្មោះមួយទៀត នឹងមាន ទីតាំងផ្ទុកដូចគ្នា។ វាត្រូវបានកំណត់ឡើងដោយបន្ថែមសញ្ញា & នៅជាប់ពីក្រោយប្រភេទទិន្នន័យរបស់អញ្ញាត។

ឧទាហរណ៏

int n = 1;

ហើយ r ជាអញ្ញាត reference ចំពោះអញ្ញាត n នោះគេអាចសរសេរដូចខាងក្រោមនេះ

int & r = n ; // r is a reference to n .

បន្ទាប់ពីកំណត់នូវឃ្លាខាងក្រោមនេះ n និង​ r មានតម្លៃដូចគ្នា​ ។ គួរកត់សម្គាល់ថា​ r មិន​ម៉ែន ចម្លងតម្លៃរបស់ n ឡើយ ប៉ុន្តែវាគ្រាន់តែជាឈ្មោះមួយទៀតរបស់ n ប៉ុណ្ណោះ។ អញ្ញាតទាំងពីរនេះមានទីតាំងផ្ទុកដូចគ្នា។

ការផ្លាស់ប្តូរតម្លៃ n នោះវានឹងនាំអោយផ្លាស់ប្តូរតម្លៃលទ្ធផលរបស់ r ដែរ និងច្រាសមកវិញ។

ឧទាហរណ៏

n = 5 ; // change both n and r.

វានឹងអោយលទ្ធផល​​ n និង r ស្មើនឹង​​​ 5 ។

ឧទាហរណ៏ទី១

// Program to demonstrate passing an argument by reference.

#include<iostream.h>

void any\_function ( int &p) {

cout << “ P is “ << p << endl;

p = 5 ;

}

void main() {

int a = 1;

cout << “ a is “ << a << endl;

any\_function( a );

cout << “ a is now “ << a << endl;

}

លទ្ធផលរបស់កម្មវីធីខាងលើគឺ

a is 1

p is 1

a is now 5

ឧទាហរណ៏ទី២

//passing two arguments by reference.

#include<iostream.h>

void swap (float val1, float val2 );

void main() {

flaot n1, n2;

cout << “ enter n1= “ ; cin >> n1;

cout << “ enter n2= “ ; cin >> n2 ;

if ( n1 > n2 )

swap( n1 , n2 );

cout << “ The numbers in order are “ << n1 << “and”<< n2 <<endl;

}

void swap ( float &val1 , float &val2 ) {

float temp = val1;

val1 = val2;

val2 = temp;

}

៩. អំពី Recursive function

Function មួយដែលមានការហៅខ្លួនវាមកប្រើដោយផ្ទាល់ ឬដោយប្រយោលហៅថា Recursive function ។ លើសពីនេះ នៅក្នុងតួនៃ function ត្រូវមានលក្ខខណ្ឌបញ្ចប់មួយដែល មិនម៉ែនជា​ recursive ។

ឧទាហរណ៏ទី​១

#include<iostream.h>

void function1 ();

void main() {

…………

…………

function1();

}

Void function1() {

…………

…………

function1();

}

ឧទាហរណ៏ទី២

// sum = 1 + 2 + 3 + ……… + n Using recursive function

#include < iostream.h>

int sum ( int n ) {

int sum (int );

int value = 0 ;

if ( n == 0 )

return (value) ;

else

value = n + sum ( n-1) ;

return (value );

}

void main () {

int n, temp;

cout << “ Enter any integer number : “ ; cin >> n;

temp = sum ( n ) ;

cout << “ value = “ << n << “ and its sum = “ << temp ;

}

លទ្ធផលរបស់កម្មវិធីខាងលើគឺ

Enter any integer number :

10

Value = 10 and its sum = 55

ឧទាហរណ៏ទី៣

//factorial of a given number using recursive function

#include< iostream.h>

long int fact (long int n ) {

long int fact ( long int );

long int value = 1 ;

if ( n== 1 ) return (value );

else {

value = n \* fact ( n – 1 ) ;

return (value) ;

}

}

void main() {

long int x , n ;

cout << “ Enter any integer number : “ << endl;

cin >> n;

x = fact ( n);

cout << “ Value = “ << n << “ and its factorial = “ << x ;

}

លទ្ធផលរបស់កម្មវិធីខាងលើគឺ

Enter any integer number :

5

Value = 5 and its factorial = 120

១០. អំពី overloaded function

នៅក្នុងផ្នែកនេះ យើងនឹងសិក្សាអំពីលក្ខណ:ពិសេសមួយដែលគួរអោយចាប់អារម្មណ៏ របស់ C++ គឺ function overloading ។ នៅក្នុងភាសា C++ កាលណាវាមាន functions ដែលមាន ឈ្មោះដូចគ្នាចាប់ពីពីរឡើងទៅ ហើយវាខុសគ្នាដោយចំនួនប៉ារ៉ាម៉ែត្រ ឬប្រភេទទិន្នន័យរបស់ ប៉ារ៉ាម៉ែត្រ និងស្ថិតក្នុង scope តែមួយនោះគេហៅវាថា overloaded function ។​ overloaded function ជាផ្នែកមួយនៃ polymorphism ។

ឧទាហរណ៏

//overload a function three times.

#include<iostream.h>

void func ( int i );

void func ( int i, int j );

void func ( double k );

void main () {

func (11) ;

func ( 11 , 22 );

func ( 3.14 );

}

void func (int i ) {

Cout << “ In func ( int ), i is “ << i << endl;

}

void func (int i , int j ) {

cout << “ In func ( int , int ) , i is << i ;

cout << “ , j is “ << j << endl;

}

void func (double k ) {

cout << “ In func (double ) , k is “ << k << ‘\n’;

}

កម្មវិធីខាងលើអោយលទ្ធផលដូចខាងក្រោម

In func (int ) i is 11

In func ( int , int ) i is 11 , j is 22

In funct (double) , k is 3.14

ដូចយើងបានឃើញហើយថា​​ func () មានចំនួនលើសដល់ទៅបីដង។ ទម្រង់ទីមួយមាន ប៉ារ៉ាម៉ែត្រមួយជាចំនួនគត់ ហើយទម្រង់ទីពីមានប៉ារ៉ាម៉ែត្រពីរជាចំនួនគត់ និងទម្រង់ទីបីមាន​ ប៉ារ៉ាម៉ែត្រមួយជាប្រភេទ double ។ ដោយសារការរាយប៉ារ៉ាម៉ែត្រចំពោះទម្រង់និមួយៗមានសភាពខុសគ្នានោះធ្វើអោយ compiler អាចហៅ function នោះមកប្រើបានត្រឹមត្រូវតាមទម្រង់ របស់វា ។

ឧទាហរណ៏

#include<iostream.h>

int myabs (int i );

double myabs ( double d );

long myabs ( long l );

void main () {

cout << myabs (-11 ) << ‘\n’;

cout << myabs (-11.0 ) << ‘\n’;

cout << myabs (-9L) << endl;

}

int myabs ( int i ) {

if ( i < 0 )

Return -i;

else

return i;

}

double myabs ( double d ) {

cout << “using double myabs() : “ ;

If ( d < 0.0 )

return -d;

else

return d;

}

long myabs ( long l ) {

cout << “using long myabs () : “ ;

if ( l < 0)

Return -l;

else

return l;

}

លទ្ធផលរបស់កម្មវីធីខាងលើគឺ

Using integer myabs () : 11

Using doyble myabs () : 11

Using long myabs () : 9

កម្មវីធីខាងលើនេះមាន function ដែលមានឈ្មោះ myabs() ចំនួនបី ហើយវាខុសគ្នាដោយប្រភេទទិន្នន័យនៃ arguments ។ ពេលគេហៅ function ទាំងនេះមកប្រើវានឹងអោយតម្លៃដាច់ខាតនៃ argument របស់វា។ compiler អាចដឹងបានតម្លៃរបស់វាដែលត្រូវគ្នានឹងប្រភេទទិន្នន័យ របស់ argument ។

១១. អំពី default arguments

មធ្យោបាយមួយដែលមានសារ:សំខាន់បំផុតនៅក្នុងភាសា C++ នោះគឺការកំណត់តម្លៃ

អោយ default arguments សម្រាប់ functions ។ នៅក្នុងលំនាំដើមនៃការប្រកាស function គេ អោយតម្លៃ default ។ នៅពេលហៅ function មកប្រើដោយគ្មានការកំណត់តម្លៃ argument នោះ​កម្មវិធីនឹងកំណត់តម្លៃទៅឲ្យប៉ារ៉ាម៉ែត្រដោយស្វ័យប្រវត្តពីលំនាំដើមនៃការប្រកាស function​ ធម្មតា ។ default arguments ជួយសម្រួលដល់ការបង្កើត និងរក្សាកម្មវីធីឲ្យមានលក្ខណ: ងាយស្រួល ។

ឧទាហរណ៏

// default argument declaration

#include < iostream.h>

void sum ( int a, int b, int c = 5, int d = 7 );

void main () {

int a, b ,c , d;

cout << “Enter any two number : “;

cin >> a >> b ;

sum ( a, b ) ;

}

void sum ( int a1, int a2 , int a3, int a4 ) {

int temp;

temp = a1 + a2 + a3 + a4;

cout << “ a = “ << a1 << endl;

cout << “ b = “ << a2 << endl;

cout << “ c = “ << a3 << endl;

cout << “ d = “ << a4 << endl;

cout << “ sum = “ << temp ;

}

កម្មវិធីខាងលើនេះត្រូវបានកែប្រែបន្តិចដើម្បីធ្វើឲ្យ sum() function ប្រើជាមួយនឹងទិន្នន័យដែល បញ្ចូលដោយអ្នកប្រើប្រាស់ទៅអោយប៉ារ៉ាម៉ែត្រ។ តាមវិធីនេះតម្លៃដើម (default values) មិនត្រូវបានកំណត់ទៅឲ្យ sum() function ឡើយ។

ឧទាហរណ៏

#include < iostream.h>

void sum ( int a, int b, int c = 15, int d = 17 );

void main () {

int a, b ,c , d;

cout << “Enter any two number : “;

cin >> a >> b ;

sum ( a, b, c , d ) ;

}

void sum ( int a1, int a2 , int a3, int a4 ) {

int temp;

temp = a1 + a2 + a3 + a4;

cout << “ a = “ << a1 << endl;

cout << “ b = “ << a2 << endl;

cout << “ c = “ << a3 << endl;

cout << “ d = “ << a4 << endl;

cout << “ sum = “ << temp ;

}

ចូរសង្កេតលើករណីពិសេសមួយចំនួននៃការប្រើលំនាំដើមរបស់ function ជាមួយនឹង default arguments ដូចខាងក្រោមនេះ

ករណីទី១

#include < iostream.h>

void sum ( int a=5 , int b, int c =7 , int d=10 );

void main() {

…………

…………

sum (b); // invalid

}

យើងនឹងឃើញថាមាន error កើតឡើងដោយវាបញ្ជាក់ប្រាប់ថា

* default value missing following parameter ‘a’ int function main()
* undefined symbol ‘b’ in function main()

ករណីទី២

#include<iostream.h>

void sum ( int a =5 , int b =7 , int c , int d );

void main () {

………….

………….

Sum ( c, d ) ; // invalid

}

យើងនឹងឃើញថា code ខាងលើក៏មិនត្រឹមត្រូវដែរ ។

ករណីទី៣

#include<iostream.h>

void sum ( int a , int b, int c =5 , int d=9 );

void main() {

………

………

Sum (a, b ); // valid

}

យើងនឹងឃើញថា code ខាងលើត្រឹមត្រូវ ។

១២. អញ្ញាត automatic

ជាអញ្ញាតដែលប្រកាសនៅខាងក្នុង function មួយ ។ វាមានលក្ខណ:ស្វ័យប្រវត្តនូវការ បង្កើតទីតាំង memory នៅពេល function ចូលប្រើ ឬឈប់ប្រើទំហំ memory នោះពេលចាក ​ ចេញ។ គេអាចនិយាយម៉្យាងទៀតថាអញ្ញាត automatic ត្រូវបានផ្តល់នូវទំហំ memory ជា បណ្តោះអាសន្នប៉ុណ្ណោះ។​ វាពុំមានអត្ថន័យឡើយនៅពេលវាប្រកាសនៅខាងក្រៅ function ។ អញ្ញាត automatic អាចប្រកាសនៅខាងដើមនៃ function ឬខាងដើមនៃ block ។ គេប្រើពាក្យ auto ដើម្បីបញ្ជាក់ប្រាប់អំពីការប្រើលក្ខណ:នេះ ប៉ុន្តែជូនកាលគេពុំប្រើវាឡើយនៅពេលប្រកាស ។

ឧទាហរណ៏

#include<iostream.h>

void main () {

auto int a, b , c ;

………………

………………

}

ការសរសេរបែបនេះ គឺវាដូចគ្នាទៅនឹងការសរសេរ

#include<iostream.h>

void main () {

int a, b , c ;

……………

}

តាមពិតគេប្រើពាក្យ auto គ្រាន់តែសរសេរដើម្បីបញ្ជាក់ឲ្យឃើញច្បាស់ថាអញ្ញាតនេះ មានលក្ខណ:​ automatic ។

//automatic variable declaration

#include<iostream.h>

void funct (int i );

void main () {

int i = 1;

funct (5);

cout << “ i = “ << i << “ (in function main ) “ << endl;

}

void funct ( int i ) {

i = i \* 10;

{

int i ;

i =2 ;

cout << “ i = “ << i << “ (in inner most block ) “ ;

cout << endl;

}

cout << “ i = “ << i << “ ( parameter of function funct) “ <<endl;

}

លទ្ធផលរបស់កម្មវិធីខាងលើគឺ

i = 2 ( in inner most block)

i = 50 (parameter of function funct )

i = 1 (in function main() )

អញ្ញាត automatic មានសារ:សំខាន់ចំនួនពីរដែលគួរឲ្យកត់សំគាល់ ។ សារ:សំខាន់ ទីមួយទំហំ memory ត្រូវបានប្រើដោយសំចៃ ពីព្រោះវាប្រើតាមដែលវាត្រូវការប៉ណ្ណោះ ។ សារ: សំខាន់ទីពីរគឺទំហំ scope របស់វាតូចអាចការពារអំពីការចូលប្រើមិនត្រឹមត្រូវរបស់ function ផ្សេងទៀត។ ហេតុនេះហើយ អញ្ញាតនៅក្នុង functions ផ្សេងទៀតមិនចាំបាច់ផ្តល់ឈ្មោះខុស គ្នាឡើយ។

១៣. អញ្ញាត register

អញ្ញាត automatic ត្រូវបានផ្ទុកក្នុង memory ។ ការចូលទៅប្រើទីតាំង memory ត្រូវ​ ចំណាយពេលច្រើន (ច្រើនជាងការចូលទៅប្រើ register របស់ម៉ាស៊ីន) គេអាចធ្វើអោយកុំព្យូទ័រ រក្សាបាននូវចំនួន អញ្ញាតមួយចំនួនតូចក្នុង register របស់វាសម្រាប់ដំណើរការលឿន។

ទំរង់ទូទៅរបស់វាគឺ

register datatype var1, var2 , … , varN;

ក្នុងបរិមាណកំណត់នៃអញ្ញាត register អាចធ្វើឲ្យកុំព្យូទ័ររក្សាបានក្នុង registers លឿនជា អចិន្ត្រៃយ៏។​ បើសិនវាពុំមានលក្ខណ:គ្រប់គ្រាន់សម្រាប់អញ្ញាត register ទេនោះសំណើប្រើ register នឹងត្រូវអសាបង់។

​​​​​//register variable declaration

#include<iostream.h>

int funct ( register int x , register int y );

void main () {

register int x, y, z ;

x = 0 ; y = 0 ;

cout << “ x “ << ‘\t’ << “y” << endl;

cout << “ --------------- “ << endl;

do {

z = funct ( x, y ) ;

cout << x << ‘\t’ << z << endl;

++x;

++y;

}while ( x < =5 );

}

int funct ( register int x , register int y ) {

register int temp ;

temp = x \* y ;

return (temp );

}

១៤. អញ្ញាត static

អញ្ញាត static ត្រូវកំណត់នៅក្នុង function មួយហើយវាមានទំហំ scope ដូចគ្នានឹង អញ្ញាត automatic ដែរ​ ក៏ប៉ុន្តែនៅក្នុងករណីអញ្ញាត static តម្លៃរបស់អញ្ញាតរក្សារហូតក្នុងកម្ម វិធីរបស់វា។ វាមានទម្រង់ទូទៅដូចខាងក្រោម

static datatype var1, var2, …. varN ;

ឧទាហរណ៏

static int x , y ;

static int x = 55;

static char a, ch;

// using the automatic variable

#include < iostream.h>

int funct ( int x );

void main () {

int i , value;

for ( i = 1 ; i < = 5 ; i ++ ) {

value = funct ( i );

cout << i << ‘\t’ << value << endl;

}

}

funct ( int x ) {

int sum =50;

sum += x ;

return (sum);

}

លទ្ធផលរបស់កម្មវិធីខាងលើគឺ

1. 51
2. 52
3. 53
4. 54
5. 55

//using the static variable

#include < iostream.h>

int funct ( int x );

void main () {

int i , value;

for ( i = 1 ; i < = 5 ; i ++ ) {

value = funct ( i );

cout << i << ‘\t’ << value << endl;

}

}

funct ( int x ) {

static int sum =50;

sum += x ;

return (sum);

}

លទ្ធផលរបស់កម្មវិធីខាងលើគឺ

1. 51
2. 53
3. 56
4. 60
5. 65

១៥. អញ្ញាតក្រៅ (External variable)

ពាក្យ extern ប្រើសំរាប់ដាក់ពីមុខអញ្ញាត ឬ​ function មួយដើម្បីបញ្ជាក់ប្រាប់ឲ្យដឹងថា អញ្ញាតនោះបានកំណត់នៅខាងក្រៅ file កំពុងប្រើក៏ប៉ុន្តែវាត្រូវបានប្រើនៅក្នុង file នេះ។

ទម្រង់ទូទៅរបស់វាគឺ

Extern datatype var1, var2, …. varN ;

ឧទាហរណ៏

extern int x , y;

extern float p, q , r;

extern char a, ch;

កម្មវិធីខាងក្រោមនេះកំណត់អញ្ញាតមួយជាអញ្ញាតក្រៅ ហើយនិង function មួយដែបានកំណត់នៅខាងក្រៅ file ឈ្មោះ maindm.cpp ។ អញ្ញាតនិង function ទាំងនោះបាន កំណត់នៅក្នុង file មួយផ្សេងឈ្មោះ count.cpp ។

File : maindm.cpp

#include<iostream.h>

#include<count.cpp>

extern int couter;

extern void inc\_counter ();

void main() {

int i;

for ( i = 0 ; i < 5; i ++)

inc\_counter();

cout << “counter is “ << counter << ‘\n’ ;

}

File : count.cpp

int counter = 0 ;

void inc\_counter () {

++counter ;

}