บทที่ 5
ฟังก์ชัน (Function)

◆ ความหมายและประโยชน์ของฟังก์ชัน

ฟังก์ชัน (Function) หมายถึง ประโยคคำสั่ง(statements) ชุดหนึ่งซึ่งมีชื่อเรียกใช้โดยเฉพาะ ฟังก์ชันหนึ่ง ๆ จะทำหน้าที่เฉพาะอย่างใดอย่างหนึ่ง ส่วนอื่น ๆ ของโปรแกรมสามารถเรียกสเตตเมนต์ชุดนี้ได้ โดยการเรียกใช้ ***ชื่อฟังก์ชัน*** นั้น ๆ ให้ถูกต้องตามรูปแบบที่ฟังก์ชันนั้น ๆ กำหนดไว้

ฟังก์ชันในภาษา C++ มี 2 ประเภท คือ

1. User Defined Function คือ ฟังก์ชันที่ผู้เขียนโปรแกรมสร้างขึ้นหรือกำหนดขึ้นเอง ตามรูปแบบการสร้างฟังก์ชันของ C++ เพื่อนำมาใช้ในการเขียนโปรแกรมของตนเอง (จะกล่าวถึงรายละเอียดต่อไป)

2. Standard Function คือ ฟังก์ชันมาตรฐานที่บริษัทผู้สร้าง Compiler ภาษา C++ ได้สร้างรวบรวมไว้ในคลัง (Library) ผู้เขียนโปรแกรมสามารถเรียกใช้ได้ทันที ได้แก่ ฟังก์ชันที่ประกอบอยู่ใน header file ต่าง ๆ ขณะเรียกใช้ ต้อง #include ชื่อไฟล์ที่รวบรวมฟังก์ชันนั้นไว้ก่อน เช่น ฟังก์ชัน clrscr() ทำหน้าที่ล้างจอภาพให้ว่างและเลื่อน cursor ไปไว้ที่มุมซ้ายบนของจอภาพ เป็นฟังก์ชันอยู่ในไฟล์ conio.h เป็นต้น

ประโยชน์ฟังก์ชันในการเขียนโปรแกรมใน C++ มีดังต่อไปนี้

1. ช่วยให้ไม่ต้องเขียน statement เดิม ๆ ซ้ำกันหลายครั้งในโปรแกรมเดียวกัน
2. ช่วยให้สามารถค้นหาส่วนที่ผลหรือส่วนที่ต้องปรับปรุงได้อย่างรวดเร็ว เนื่องจากเราทราบว่าแต่ละฟังก์ชันทำหน้าที่อะไร และสร้างฟังก์ชันไว้ที่ส่วนใดของโปรแกรม
3. ทำให้โปรแกรมมีขนาดกระทัดรัด ทำความเข้าใจได้ง่ายและรวดเร็ว เพราะเขียนแบ่งเป็นฟังก์ชันตามงานหรือหน้าที่ที่ต้องการเขียน
4. เนื่องจากฟังก์ชันมีการทำงานเป็นอิสระ สามารถนำฟังก์ชันที่สร้างไว้และมีการทำงานเป็นมาตรฐานแล้ว เก็บไว้ใน Header File เพื่อให้เป็นคลังคำสั่ง(Library) ของผู้ใช้ นำไปใช้ได้ในโปรแกรมอื่น ๆ ได้อีก ซึ่งช่วยให้เราเขียนโปรแกรมใหม่ได้รวดเร็วขึ้น โดยการนำฟังก์ชันที่มีอยู่แล้วมาใช้ใหม่ได้

♦ **ส่วนประกอบในการสร้างฟังก์ชัน**

ฟังก์ชันที่เราสร้างขึ้นเอง เรียกว่า ***User Defined Function*** ซึ่งใน C++ กำหนดให้เมื่อมีการเรียกใช้ชื่อฟังก์ชันใด ๆ จะต้องมีการ ***ส่งค่ากลับมา (return value)*** ในชื่อฟังก์ชันนั้นเสมอ มีรูปแบบโครงสร้างของฟังก์ชันประกอบไปด้วยส่วนหัวของฟังก์ชันและส่วนของ statement ดังนี้

```
function_type  function_name (parameter1,parameter2,…)  //function header
{  int  …          //declaration variable in function
   float  …                //declaration variable in function
      statement;                    //statement in function
      statement;                    //statement in function
      return(value);      //function must return only one or null value to function_name
}
```

ความหมายของส่วนประกอบในฟังก์ชัน มีดังนี้

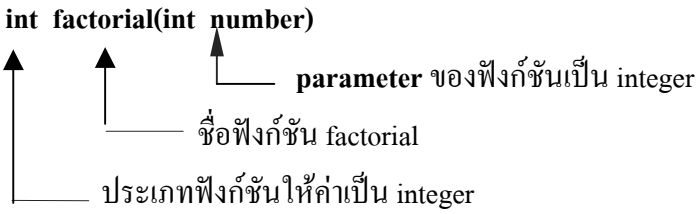
function_type คือ ประเภทค่าหรือข้อมูลที่ได้จากการทำงานของฟังก์ชัน ซึ่งจะต้องให้ค่าคืนกลับมาเก็บไว้ในชื่อของฟังก์ชัน (function_name) ถ้าเป็นประเภท void จะเป็นฟังก์ชันประเภทที่ต้องมีการ return value

function_name คือ ชื่อฟังก์ชันที่กำหนดขึ้นตามกฎเกณฑ์การตั้งชื่อของ C++ และจะเป็นชื่อที่ใช้สำหรับการเรียกใช้ฟังก์ชันต่อไป

(parameter1,parameter2,…) หรือพารามิเตอร์ประกอบไปด้วย ***ประเภทและชื่อของตัวแปร*** ใช้รับค่าคงที่เพื่อนำมาใช้ทำงานในฟังก์ชัน ในขณะที่ฟังก์ชันนั้นถูกเรียกใช้ทำงาน พารามิเตอร์ของฟังก์ชันมีมากกว่า 1 ตัวและหลายประเภทได้

return(value); โดยที่ return เป็นคีย์เวิร์ด และ value คือค่าคงที่ที่ส่งคืนไปให้แก่ชื่อฟังก์ชัน 1 ค่าหรือไม่มีก็ได้ (กรณีเป็นฟังก์ชันที่ไม่ให้ค่าใด ๆ)

ตัวอย่างการกำหนดชื่อฟังก์ชัน เช่น



ความหมาย คือ ฟังก์ชันชื่อ factorial() ค่าที่ได้จากการทำงานของฟังก์ชันเป็น int และขณะเรียกใช้ต้องมีการรับค่าพารามิเตอร์มาเก็บไว้ที่ตัวแปร number และเป็นประเภท int เพื่อในมาใช้ภายในฟังก์ชัน

void line()

ความหมายคือ ฟังก์ชันชื่อ line() เป็นฟังก์ชันที่ไม่มีพารามิเตอร์ เมื่อฟังก์ชันทำงานเสร็จแล้ว จะไม่ให้ค่าใด ๆ คืนกลับมา เพราะมีประเภทเป็น void

float power(int base, int exp)

ความหมาย ฟังก์ชันชื่อ power() ค่าที่ได้จากฟังก์ชันเป็น float และขณะเรียกใช้ต้องมีการรับค่าพารามิเตอร์ 2 ตัว ตัวแรก base เป็น int และตัวที่สอง exp เป็น int มาใช้ในฟังก์ชันด้วย

♦ การสร้างและการเรียกใช้ฟังก์ชันในโปรแกรม

การสร้างและการเรียกใช้ฟังก์ชันในโปรแกรม มีการกำหนดรูปแบบการสร้างและเรียกใช้ ดังนี้

```
#include <header file>
#include <header file>
//declaration  2 prototype function in this program are line() and factorial()
void line();
int  factorial(int  number);

void main()                 //function main()
{ int   …           //declaration variable in function main()
statement;                  //statement in function
factorial(int value or variable);  //call function factorial() in function main()
   statement;               //statement in function
     line();            //call function line() in function main()
   statement;               //statement in function
} //end of main() and end of this  program

 void line()
  { int   …          //declaration variable in function
 float  …            //declaration variable in function
    statement;              //statement in function
    statement;              //statement in function
  }  //return

 int  factorial(int  number)
  { int   …          //declaration variable in function factorial()
 float  …            //declaration variable in function factorial()
    statement;              //statement in function factorial()
  line();           //call function  line()  in function factorial()
   statement;               //statement in function factorial()
    return(value);//return only one value to factorial() function
} //return
```

จากโครงสร้างโปรแกรมที่มีการสร้างฟังก์ชัน สรุปได้ดังนี้

1. การประกาศชื่อฟังก์ชันที่ผู้ใช้กำหนดไว้ที่ตอนต้นโปรแกรมก่อนฟังก์ชัน main() เรียกว่า **ฟังก์ชันต้นแบบ (prototype)** หรือการกำหนดฟังก์ชัน (function declaration) แสดงว่าในโปรแกรมมีการสร้างและเรียกใช้ฟังก์ชันเหล่านี้ ซึ่งสามารถเรียกใช้ชื่อฟังก์ชันนั้น ณ ตำแหน่งใด ๆ ของโปรแกรมก็ได้ **ยกเว้นห้าม** เรียกใช้ชื่อฟังก์ชันนั้นในตัวฟังก์ชันเอง การประกาศฟังก์ชันต้นแบบ(prototype) ที่มีพารามิเตอร์(parameters) ทำได้ 2 ลักษณะ คือ

- ในส่วนของพารามิเตอร์ของฟังก์ชัน เขียนทั้งชนิดข้อมูลและตัวพารามิเตอร์ เช่น

  int factorial(int number);

- ในส่วนของพารามิเตอร์ของฟังก์ชัน เขียนเฉพาะชนิดข้อมูล เช่น

  int factorial(int);

2. การเรียกใช้ฟังก์ชันมีรูปแบบการเรียกใช้เสมือนเป็น statement เช่น line() factorial(x) โดยที่ฟังก์ชันใดที่มีพารามิเตอร์กำหนดไว้ ต้องใส่ค่า argument ให้ถูกต้องในขณะเรียกใช้ด้วย และเรียก **x** ของฟังก์ชัน factorial() ว่าอากิวเมนต์ (**argument**)

3. เมื่อมีการเรียกใช้ฟังก์ชัน ณ ตำแหน่งใด โปรแกรมจะไปทำงานในฟังก์ชันนั้นจนเสร็จ แล้วจะกลับมาทำงานต่อใน statement ถัดไปจากตำแหน่งที่เรียกใช้ฟังก์ชัน

4. สามารถเรียกใช้ฟังก์ชันกี่ครั้งก็ได้ในโปรแกรม

♦ ฟังก์ชันไม่มีพารามิเตอร์

ฟังก์ชันไม่มีพารามิเตอร์ หมายถึง ฟังก์ชันที่ไม่มีการรับค่าพารามิเตอร์มาใช้ในขณะทำงาน และการเรียกใช้ก็ไม่ต้องส่ง argument มาให้ฟังก์ชัน อาจเป็นฟังก์ชันประเภทคืนค่า (return value) หรือไม่มีการคืนค่าใด ๆ ให้แก่ชื่อฟังก์ชันเมื่อทำงานเสร็จก็ได้

- ***ตัวอย่างโปรแกรม line_non.cpp แสดงการสร้างและเรียกใช้ฟังก์ชันแบบไม่มีพารามิเตอร์และไม่มีการคืนค่าใด ๆ คือ ฟังก์ชันประเภท viod***

```
/*Program : line_non.cpp
  Process : creat non-parameter function line() */
#include <iostream.h>
#include <conio.h>

//declaration prototype function
void line();
//begin main program
void main()
{ int i;
```

```
 clrscr();
 cout<< "Display 1 line from line()  function\n";
 line(); //call line() and non-argument
 getch();
 cout<< "Display 5 lines from line()  function\n";
 for (i=1;i<=5;++i)
   line();  //call line() and non-argument
 getch();
}//end main program

void line()  //non-parameter function
{
 cout<< "_____\n";
}
```

- ***ตัวอย่างโปรแกรม func_non.cpp แสดงโปรแกรมที่มีการสร้างฟังก์ชันประเภทที่ไม่มี parameter*** ไม่มีการ return value คือ ฟังก์ชัน input() และฟังก์ชันที่ไม่มี parameter แต่มีการ return value คือฟังก์ชัน summation()

```
/*Program : func_non.cpp
  process : display return value and non-parameter function
*/
#include <iostream.h>
#include <conio.h>
//prototype function declaration
int summation();
void input();
//global variable
int x,y;

void main()
{ clrscr();
 input();     //called function input()
 cout<<"Result of x+ y =\a"<<summation();   //called function summation()
 getch();
}


void input()   //non-return value function
{  cout<<"Enter 2 number for summation : "<<endl<<endl;
   cout<<"Number1 : ";cin>>x;
```

```
cout<<"Number2 : ";cin>>y;
}

int summation()   //return value function
{ int result;
  result=x+y;
  return (result);  //return result value to summation() called
}
```

♦ ฟังก์ชันมีพารามิเตอร์

ฟังก์ชันมีพารามิเตอร์ หมายถึงฟังก์ชันที่มีการรับค่าพารามิเตอร์มาใช้ในขณะฟังก์ชันทำงาน ดังนั้นการเรียกใช้ฟังก์ชันประเภทนี้ ต้องส่ง argument มาให้ฟังก์ชันด้วย จึงจะสามารถทำงานได้ ดังตัวอย่าง

- ***ตัวอย่างโปรแกรม line_par.cpp แสดงการสร้างและเรียกใช้ฟังก์ชันแบบมีพารามิเตอร์และไม่มีการคืนค่าใด ๆ (เป็นฟังก์ชันประเภท void) ให้แก่ชื่อฟังก์ชัน ณ จุดเรียกใช้ ค่าที่ได้จากการทำงานของฟังก์ชัน จะต้องนำไปใช้อย่างใดอย่างหนึ่งในโปรแกรม เช่น แสดงค่านั้นทางจอภาพ เก็บค่าที่ได้ไว้ในตัวแปร เป็นต้น***

```
/*Program : line_par.cpp
  Process : creat parameter function line()*/
#include <iostream.h>
#include <conio.h>
void line(int amount);  //declaration prototype function
//begin main program
void main()
{ int i=45;
  //begin statement
  clrscr();
  cout<< "Display line from line() function\n";
  line(30); //call line() and send constant argument is 30
  getch();
  line(50); //call line() and send constant argument is 50
  getch();
  line(i);  //call line() and send variable argument is i
  getch();
}
//end main program
void line(int amount)  //amount is int parameter of function
{
  int x;
```

```
 for(x=1; x<=amount;++x)
   cout<<"_";
 cout<<'\n';
}
```

หมายเหตุ จากโปรแกรมตัวอย่าง การกำหนดค่าอาร์กิวเมนต์ (argument) ในการเรียกใช้ฟังก์ชันทำได้ 2 ลักษณะ คือ

1. กำหนดเป็นค่าคงที่ (constant) ให้เป็นประเภทเดียวกันกับพารามิเตอร์ของฟังก์ชัน เช่น การเรียกใช้ฟังก์ชัน line(30) line(50) ซึ่ง 30 และ 50 คือ ค่าคงที่ชนิด int

2. กำหนดเป็นตัวแปร(variable) โดยกำหนดให้เป็นตัวแปรประเภทเดียวกันกับพารามิเตอร์ของฟังก์ชัน เช่น การเรียกใช้ฟังก์ชัน line(i) ซึ่ง i เป็นตัวแปรชนิด int

- ***ตัวอย่างโปรแกรม func_par.cpp แสดงการคำนวณการบวก ลบ คูณ หาร เลขจำนวนจริง 2 จำนวน โดยการสร้างฟังก์ชันประเภทที่มี parameters และมีการ return value ให้แก่ชื่อฟังก์ชัน ณ ตำแหน่งที่เรียกใช้ฟังก์ชัน***

```
/*Program : func_par.cpp
  process : display return value and parameter function
          calculate addition,subtract,multiply and divide
*/
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
//prototype function declaration
void input();
float addition(float x, float y);
float subtract(float x, float y);
float multiply(float x, float y);
float divide(float x, float y);
//global variable
float num1,num2;

void main()
{  clrscr();
   input();
  cout<<"Result of addition = "<<addition(num1,num2)<<endl;
  cout<<"Result of subtract = "<<subtract(num1,num2)<<endl;
  cout<<"Result of multiply = "<<multiply(num1,num2)<<endl;
  cout<<"Result of subtract = "<<divide(num1,num2)<<endl<<endl;
  cout<<"Calculate by sent argument to parameters of function..."<<endl;
```

```
 cout<<"Result of  10.0+20.25= "<<addition(120.0,20.25)<<endl;
 cout<<"Result of  10.0-20.25= "<<subtract(120.0,20.25)<<endl;
 cout<<"Result of  10.0*20.25= "<<multiply(120.0,20.25)<<endl;
 cout<<"Result of  10.0/20.25= "<<setprecision(3)<<divide(10.0,20.25)<<endl;
 getch();
}

void input()   //non-parameter and non-return value function
{
  cout<<"Enter 2 number for calculate : "<<endl<<endl;
  cout<<"Number1 : ";cin>>num1;
  cout<<"Number2 : ";cin>>num2;
}

float addition(float x, float y)
{ float result;
 result=x+y;
result=x-y;
 return (result);
}

float subtract(float x, float y)
{ float result;
  result=x-y;
  return (result);
}

float multiply(float x, float y)
{ float result;
 result=x*y;
 return (result);
}

float divide(float x, float y)
{ float result;
 result=x/y;
 return (result);
}
```

♦ ฟังก์ชันแบบไม่ประกาศ prototype

การสร้างฟังก์ชันในโปรแกรมแบบไม่ประกาศเป็น prototype คือ การสร้างรายะเอียดของฟังก์ชันต่างๆ ไว้ก่อนฟังก์ชัน **main()** การเรียกใช้ฟังก์ชันประเภทนี้จะต้องเรียกใช้ตามลำดับของการสร้างฟังก์ชันในโปรแกรม ฟังก์ชันที่ถูกเรียกใช้จะต้องสร้างอยู่ก่อนฟังก์ชันที่เรียกใช้เสมอ เหมาะสำหรับการเขียนโปรแกรมง่าย ๆ หรือมีขนาดสั้น ๆ ไม่สลับซับซ้อนมากนัก

- ***ตัวอย่างโปรแกรม non_prot.cpp แสดงการสร้างฟังก์ชันแบบไม่ประกาศ prototype ไว้ก่อน ดังรายละเอียดโปรแกรมในหน้าต่อไป***

```
/*Program : non_prot.cpp
  Process : creat non prototype function */
#include <iostream.h>
#include <conio.h>

// create non-prototype function before function main()
void line1(int amount)  //parameter function
{
  int x;
  for(x=1; x<=amount;++x)
    cout<<"_";
  cout<<'\n';
}

void line2() //non-parameter function
{
  cout<< "**********************************\n";
  line1(65);
}
//begin main program
void main()
{
  clrscr();
  cout<< "Display line from line1() function\n";
  line1(40);          //call line1() and constant argument is 40
  getch();
  cout<< "\nDisplay line from line2() function\n";
  line2();            //call line2() and constant argument is 50
  getch();
} //end main program
```

♦ อินไลน์ฟังก์ชัน (inline function)

โปรแกรมที่เรียกใช้ฟังก์ชันจะมีการเรียกใช้หน่วยความจำน้อยกว่าโปรแกรมที่ไม่เรียกใช้ฟังก์ชัน ทั้งนี้เพราะฟังก์ชันจะจองและใช้พื้นที่หน่วยความจำขณะถูกเรียกใช้เท่านั้น เมื่อฟังก์ชันทำงานเสร็จจะคืนพื้นที่หน่วยความจำให้กับโปรแกรม ส่วนโปรแกรมที่ไม่เรียกใช้ฟังก์ชันจะมีการใช้หน่วยความจำเต็มที่ตลอดเวลาตามความต้องการของโปรแกรมในขณะทำงาน

โปรแกรมที่เรียกใช้ฟังก์ชันมีส่วนดีในแง่ประหยัดการใช้หน่วยความจำ แต่การเรียกใช้ฟังก์ชันทำให้โปรแกรมต้องใช้เวลาเพิ่ม มีผลทำให้การทำงานของโปรแกรมช้ากว่าโปรแกรมที่ไม่เรียกใช้ฟังก์ชัน แต่เราสามารถกำหนดให้ฟังก์ชันในโปรแกรมถูกเรียกใช้โดยจองพื้นที่หน่วยความจำตลอดเวลา ไม่เสียเวลาในการเรียกใช้ โดยกำหนดให้ฟังก์ชันเป็นประเภท อินไลน์ฟังก์ชัน (inline function)

วิธีการสร้าง อินไลน์ฟังก์ชัน (inline function) มีวิธีการดังนี้

1. สร้างอินไลน์ฟังก์ชันอยู่ก่อนฟังก์ชัน main()
2. เพิ่มคีย์เวิร์ดคำว่า inline ที่ส่วนหัวของฟังก์ชัน

หมายเหตุ ข้อจำกัดของอินไลน์ฟังก์ชัน คือ ฟังก์ชันต้องมีขนาดสั้น ถ้าอินไลน์ฟังก์ชันมีขนาดยาวเกินไป compiler ของ C++ จะ compile ให้เป็นฟังก์ชันที่มีการเรียกใช้แบบปกติ

- ***ตัวอย่างโปรแกรม inline.cpp แสดงการสร้างและการเรียกใช้ อินไลน์ฟังก์ชัน ชื่อ square() ซึ่งทำหน้าที่คำนวณหาค่าเลขยกกำลังสอง ดังนี้***

```
/*Program : inline.cpp
 Process : creat and call inline function*/

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

 inline float square(float number) //declaration inline function
 {
   return(number*number);
 }

 void main() //begin function main()
 { float x;
  clrscr();
  cout<< "Enter number to calculate square : ";
  cin>>x;
  cout<< "Result of square = "<<square(x);  //call square() function
  getch();
 }
```

◆ รูปแบบการส่งอาร์กิวเมนต์ให้ฟังก์ชัน (reference argument)

ฟังก์ชันประเภทที่สร้างแบบมีพารามิเตอร์(parameter) เมื่อถูกเรียกใช้จะต้องมีการส่งอาร์กิวเมนต์ (argument) ไปให้แก่ฟังก์ชันนั้น ๆ โดยที่ชนิดของตัวแปรหรือค่าคงที่ที่ของอาร์กิวเมนต์จะต้องเป็นชนิดเดียวกับพารามิเตอร์ของฟังก์ชันและมีจำนวนเท่ากับพารามิเตอร์ และกรณีที่มีพารามิเตอร์มากกว่า 1 ตัว ลำดับการส่งอาร์กิวเมนต์ต้องตรงกับลำดับของพารามิเตอร์ที่กำหนดไว้ ดังตัวอย่าง เช่น

```
void main()
{  int  x = 5, y=2;
   float  result;
   result =  power(x,  y);  //เรียกใช้ฟังก์ชัน  power() ต้องส่ง argument x และ y ไปให้ฟังก์ชัน
}                           โดยที่  x และ  y  เป็นตัวแปรประเภท int เหมือนกับตัว parameters

float  power(int  base,  int  exp)  //ตัวแปร base , exp เป็น parameters ของ function
{                                   //รอรับค่าจาก argument x และ y ตามลำดับ
  statement1;
  statement2;
  return   value;
}
```

ลักษณะของการส่งอาร์กิวเมนต์ที่เป็น ***ตัวแปร*** ให้แก่ฟังก์ชัน มี 2 ลักษณะ คือ

1. ส่งเฉพาะค่าอาร์กิวเมนต์ (passed argument by value) หมายถึง เมื่อส่งค่าของตัวแปรที่เป็นอาร์กิวเมนต์ไปให้แก่พารามิเตอร์ของฟังก์ชันแล้ว เมื่อฟังก์ชันทำงานเสร็จแล้ว ค่าตัวแปรที่เป็นอาร์กิวเมนต์ยังคงมีค่าเดิม การทำงานของฟังก์ชันไม่มีผลทำให้ตัวแปรอาร์กิวเมนต์เปลี่ยนแปลงค่า เรียกอาร์กิวเมนต์ประเภทนี้ว่า **value argument**

2. ส่งและเปลี่ยนค่าอาร์กิวเมนต์(passed argument by reference) เมื่อส่งค่าของตัวแปรที่เป็นอาร์กิวเมนต์ไปให้แก่พารามิเตอร์ของฟังก์ชันแล้ว เมื่อฟังก์ชันทำงานเสร็จแล้ว ค่าตัวแปรที่เป็นอาร์กิวเมนต์มีการเปลี่ยนแปลง การทำงานของฟังก์ชันทำให้ตัวแปรอาร์กิวเมนต์เปลี่ยนแปลงค่าไปจากเดิม เรียกอาร์กิวเมนต์ชนิดนี้ว่า **reference argument**

รูปแบบการกำหนดและการเรียกใช้ฟังก์ชันที่มีการส่งอาร์กิวเมนต์แบบ value argument และ reference argument มีดังนี้

```
#include <header file>
//declaration prototype function
float  square(float number);
float area(float&  width,  float&  high);

void main()
{  float  x,y,z;
   statement;
 }
float square(float number)   // value argument   function
{
   statement;
    return value_of_function;
}
float area(float&  width,  float&  high)  //  reference argument function
{
    statement;
    return value_of_function;
}
```

หมายเหตุ การกำหนดฟังก์ชันให้เป็นประเภท reference argument ใช้เครื่องหมาย & ต่อท้ายชนิดข้อมูลของพารามิเตอร์ในการกำหนดส่วนหัวของฟังก์ชัน ในฟังก์ชันสามารถกำหนดทั้ง reference argument และ value argument ร่วมกันได้

- ***ตัวอย่างโปรแกรม argument.cpp แสดงการเปรียบเทียบการเรียกใช้ฟังก์ชันแบบ value argument และ แบบ reference argument***

```
/*Program : argument.cpp
  Process : show creat and call value and reference function*/

  #include <iostream.h>
  #include <conio.h>
  //declaration prototype function
  float sum(float arg1,float arg2);
  float multiply(float& arg3,float& arg4);

  void main()  //begin main program
  {
    float arg1,arg2,arg3,arg4;
    clrscr();
    cout<< "Please enter 4 number for argument : \n";
    cout<< "Argument1  : ";cin>>arg1;
    cout<< "Argument2  : ";cin>>arg2;
    cout<< "Argument3  : ";cin>>arg3;
    cout<< "Argument4  : ";cin>>arg4;
    //display call value argument function
    cout<< "\n\nDisplay call function sum() and send value argument";
    cout<< "\nValue argument before send to function :";
   cout<< "\nArgument 1 = "<<arg1<< "  Argument 2 ="<<arg2;
   cout<< "\nResult of sum = "<<sum(arg1,arg2);
   cout<< "\nValue argument after send to function :(not change)";
   cout<< "\nArgument 1 = "<<arg1<< "  Argument 2 ="<<arg2;
   // display call reference argument function
   cout<< "\n\nDisplay call function multiply() and send reference argument";
   cout<< "\nValue argument before send to function :";
   cout<< "\nArgument 3 = "<<arg3<< "  Argument 4 ="<<arg4;
   cout<< "\nResult of multiply from function= "<<multiply(arg3,arg4);
   cout<< "\nValue argument after send to function :(changed)";
   cout<< "\n\aArgument 3 = "<<arg3<< "  Argument 4 ="<<arg4;
   getch();
  }  //end main program

  float sum(float para1, float para2)  //value argument function
  {
    return para1+para2;
  }
float multiply(float& para3,  float& para4) //reference argument function
```

```
{
  para3++;  //change para3 and return valut to arg3
  para4++;  //change para4 and return valut to arg4
  return para3*para4;
}
```

♦ Overloaded function

โอเวอร์โหลดฟังก์ชัน(overloaded function) เป็นฟังก์ชันที่สามารถทำงานได้หลาย ๆ ลักษณะขึ้นอยู่กับอาร์กิวเมนต์ที่ผู้ใช้กำหนดให้ ดังนั้นชื่อฟังก์ชันประเภทนี้ ขณะเรียกใช้จึงสามารถกำหนดอาร์กิวเมนต์ได้หลายลักษณะ

วิธีการสร้าง overloaded function มีวิธีการสร้างเหมือนกับฟังก์ชันปกติ แต่มีการสร้างชื่อ overloaded function ให้มีชื่อเหมือนกัน โดยกำหนดพารามิเตอร์แต่ละฟังก์ชันให้แตกต่างกัน ดังตัวอย่างในโปรแกรม

- ***โปรแกรม over_fun.cpp แสดงการสร้าง overloaded function ชื่อฟังก์ชัน line()***

```
/*Program : over_fun.cpp
  Process : creat and call overloaded function line()*/
#include <iostream.h>
#include <conio.h>
//declaration overloaded prototype function
void line();
void line(int x);
void line(char ch,int x);
void line(char ch);
void main()  //begin main program
{
  clrscr();
  cout<< "Display 1 line from line()  function\n";
  line();                    //call line()
  cout<< "Display 2 line from line()  function\n";
  line(30);                  //call line()
  cout<< "Display 3 line from line()  function\n";
  line('*',60);              //call line()
cout<< "Display 4 line from line()  function\n";
  line('+');                 //call line()
  getch();
} //end main program
void line() //overload function1 line()
```

```
{
  cout<< "================================\n";
}

void line(int x)  //overload function2 line()
{ int y;
 for (y=1;y<=x;++y)
    cout<< "_";
 cout<<'\n';
}

void line(char ch,int x)  //overload function3 line()
{ int y;
 for (y=1;y<=x;++y)
    cout<< ch;
 cout<<'\n';
}

void line(char ch)  //overload function4 line()
{ int y;
 for (y=1;y<=80;++y)
    cout<< ch;
 cout<<'\n';
}
```

♦ Default argument

การประกาศ prototype function สามารถกำหนดค่าเริ่มต้น (default) ให้กับฟังก์ชันได้ ในขณะเรียกใช้ฟังก์ชันทำให้เราสามารถกำหนดอาร์กิวเมนต์ได้หลายลักษณะ ถึงแม้เราไม่กำหนด หรือกำหนดอาร์กิวเมนต์เพียงบางส่วน ฟังก์ชันก็ยังสามารถทำงานได้ เรียกอาร์กิวเมนต์แบบนี้ว่า **default argument**

- ***ตัวอย่างโปรแกรม default.cpp แสดงการสร้างฟังก์ชันแบบกำหนด default argument ไว้สำหรับการเรียกใช้ฟังก์ชัน***

```
/*Program : argument.cpp
```

```
 Process : creat and call default argument function line()*/
#include <iostream.h>
#include <conio.h>

//ประกาศ prototype function และกำหนด default argument
void line(char ch= '_',int x=80);

void main() //begin main program
{
 clrscr();
 cout<< "Display line from line()  function\n";
 line();              //call line() function
 line('+');           //call line()  function
 line('#',50);        //call line() function
 line(65);            //call line() will display character of ASCII CODE 65
 getch();
}//end main program

void line(char ch,int x)  //function  line()
{
 for (int y=1;y<=x;++y)
    cout<< ch;
 cout<<'\n';
}
```

♦ ลักษณะการใช้ตัวแปรในฟังก์ชัน

การใช้หรือการประกาศตัวแปรในโปรแกรมของ C++ สามารถกำหนดให้ตัวแปรมีลักษณะการใช้งานได้ 3 ลักษณะ คือ

1. automatic variable เป็นการประกาศตัวแปร อยู่ใน { } ของฟังก์ชันใดฟังก์ชันหนึ่ง ตัวแปรประเภทนี้จะนำไปใช้ได้เฉพาะในฟังก์ชันนี้เท่านั้น จึงเรียกอีกอย่างหนึ่งว่า Local variable การทำงานของตัวแปรประเภทนี้ จะจองพื้นที่หน่วยความจำเมื่อฟังก์ชันถูกเรียกใช้เท่านั้น และยกเลิกการใช้หน่วยความจำเมื่อฟังก์ชันทำงานเสร็จ ปกติ C++ จะไม่กำหนดค่าเริ่มต้นให้แก่ตัวแปร automatic variable ดังนั้นผู้ใช้ควรกำหนดค่าเริ่มต้นให้กับตัวแปรด้วยเพื่อป้องกันความผิดพลาด การกำหนดตัวแปร automatic variable จะใช้คีย์เวิร์ด **auto** หรือไม่ก็ได้ เช่น

```
void  main()
```

```
    { int x=50;                 //declaration x and number are automatic variable in main()
  auto float number = 0;  //used keywoad  auto  before type and name fo variable
  statement;
        …
}

    void repeat()  //function repeat()
    {    int y;         //declaration y is automatic variable
         for(y=1;y<=40;++y)  //use y from automatic or local variable
              cout<<'#';
         cout<<endl;
   }
```

2. external variable เป็นการประกาศตัวแปร อยู่นอก { } ของทุกฟังก์ชัน ตัวแปรประเภทนี้จะนำไปใช้ได้ในทุกฟังก์ชัน จึงเรียกอีกอย่างหนึ่งว่า Global variable การทำงานของตัวแปรประเภทนี้จะจองพื้นที่หน่วยความจำตลอดเวลาที่โปรแกรมทำงาน และยกเลิกการใช้หน่วยความจำเมื่อโปรแกรมทำงานเสร็จ ปกติ C++ จะไม่กำหนดค่าเริ่มต้นให้อัตโนมัติแต่อาจจะไม่ถูกต้อง ดังนั้นผู้ใช้ควรกำหนดค่าเริ่มต้นให้กับ
ตัวแปรด้วยเพื่อป้องกันความผิดพลาด การกำหนดตัวแปร external variable จะกำหนดหรือประกาศไว้ก่อนฟังก์ชัน main()

```
    int  y=0;    //declaration y and x  are external variable
    float x=0;

    void  main()  //function main()
    { int x=50;       //declaration x and number are automatic variable in main()
      statement;
      …
}
```

<u>หมายเหตุ</u> ชื่อตัวแปรประเภท automatic และ external สามารถกำหนดชื่อซ้ำกันได้ เนื่องจากใช้หน่วยความจำพื้นที่คนละส่วนกัน

3. static variable เป็นการประกาศตัวแปร อยู่ใน { } ของฟังก์ชันใดฟังก์ชันหนึ่ง ตัวแปรประเภทนี้จะนำไปใช้ได้เฉพาะในฟังก์ชันนี้เท่านั้น การทำงานของตัวแปรประเภทนี้ จะจองพื้นที่หน่วยความจำไว้ตลอดเวลาที่โปรแกรมทำงาน และยกเลิกการใช้หน่วยความจำโปรแกรมจบการทำงาน เรียกอีกอย่างหนึ่งว่า **static automatic variable** จะใช้คีย์เวิร์ด **static** นำหน้าในการประกาศตัวแปร เช่น

```
    float Avg(float value)
```

```
{  static  float total;     //declaration static variable in function Avg()
 static  int  amount;
    total = total+value;  //summation          of   total
    statement;
 }
```

- ***ตัวอย่างโปรแกรม type_var.cpp แสดงการกำหนดและการใช้ตัวแปรประเภท automatic และ external variable ในโปรแกรม***

```
/*Program : type_var.cpp
  Process : show using automatic and external variable
*/
#include <iostream.h>
#include <conio.h>

void line();  //declaration prototype function
void repeat();
int x;   //declaration external or global variable

void main()   //begin main program
{ int x;          //declaration automatic or global variable in main()
  auto int y;   //declaration automatic or global variable in main()
  clrscr();
  line();
  repeat();
  for(y=1;y<=5;++y) //use y is automatic or local variable
       line();
  for(x=1;x<=5;++x) //use x is automatic or local variable
       repeat();
getch();
} //end main program

void line()    //function line()
{ for(x=1;x<=80;x++)  //use x from global or external variable
  cout<<'=';
  cout<<endl;
 }


void repeat()  //function repeat()
{int y;
```

```
 for(y=1;y<=40;++y)  //usy y from automatic or local variable
   cout<<'#';
 cout<<endl;
}
```

♦ Template Function

เท็มเพลตฟังก์ชัน (Template Function) หมายถึง ฟังก์ชันที่มีคุณสมบัติในการใช้อากิวเมนต์ (argument) ได้กับข้อมูลหลาย ๆ ประเภท เท็มเพลตฟังก์ชันกำหนดให้พารามิเตอร์ของฟังก์ชันสามารถรับค่าอาร์กิวเมนต์ได้หลายประเภท เช่น ฟังก์ชัน sum() กำหนดให้เป็นประเภท template function สามารถรับค่าจากตัวแปรประเภท int, float และ double ได้ คำนวณมาหาผลรวมได้ทั้งอาร์กิวเมนต์ประเภท int, float และ double ซึ่งในฟังก์ชันแบบปกติพารามิเตอร์จะรับอาร์กิวเมนต์ได้เฉพาะข้อมูลประเภทเดียวกันกับที่กำหนดไว้เท่านั้น

การกำหนด template function ***<u>กำหนดไว้ก่อนหน้าฟังก์ชัน main()</u>*** มีรูปแบบดังนี้

```
template <class Type>
Type Sum(Type first,  Type second)  //function  sum()
{  statement;
   statement;
    return  first+second;
}
```

ส่วนประกอบใน template function มีดังนี้

1. **template** เป็นคีย์เวิร์ดเพื่อกำหนดว่า ฟังก์ชัน sum() เป็น template function

2. **<class Type> class** เป็นคีย์เวิร์ดเพื่อกำหนดชนิดข้อมูล มีชื่อว่า Type ซึ่งสามารถกำหนดชื่อได้ตามความต้องการของผู้ใช้ ตามกฎการตั้งชื่อของ C++

3. **Type Sum(Type first, Type second)** ชื่อฟังก์ชัน sum() มีประเภทเป็น Type มีพารามิเตอร์ 2 ตัว มีชนิดเป็น Type เช่นกัน

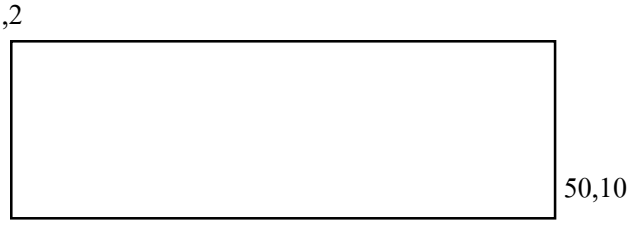
- ***ตัวอย่างโปรแกรม template.cpp แสดงการใช้ฟังก์ชันประเภท template function ชื่อ sum() ทำหน้าที่คำนวณหาผลรวมของเลข 2 จำนวน ฟังก์ชันชื่อ multiply() หาผลคูณเลข 2 จำนวน ซึ่งเป็นตัวเลขประเภท integer, float และ double***

```
/*Program : template.cpp
 Process : show create and call template function */
```

```
#include <iostream.h>
#include <conio.h>
//declaration templat function sum()
template<class TYPE>
TYPE sum(TYPE first, TYPE second)
{
  return first+second;
}
//declare template function multiply()
template<class New_type>
New_type multiply(New_type number1,New_type number2)
{
  return number1*number2;
}
void main()  //begin main program
{ int x=50,y=30;
  float a=300.25, b=100.50;
  double c=3.21541005,d=10005.02541152;
  //begin statement
  clrscr();
  cout<< " Sum integer "<< sum(x,y)<<endl;
  cout<< " Sum float "<< sum(a,b)<<endl;
  cout<< " Sum double "<< sum(c,d)<<endl;
  cout<< "\n Multiply integer "<< multiply(x,y)<<endl;
  cout<< " Multiply float "<< multiply(a,b)<<endl;
  cout<< " Multiply double "<< multiply(c,d);
  getch();
} //end main program
```

◆ แบบฝึกหัดท้ายบท

1. ให้เขียนโปรแกรมสร้างฟังก์ชันเส้นกรอบสี่เหลี่ยม โดยกำหนดพารามิเตอร์กำหนดตำแหน่ง มุมซ้ายด้านบนและมุมขวาด้านล่างของกรอบได้ เช่น frame(5,2,50,10) แสดงว่ากรอบสี่เหลี่ยมจะเริ่มวาดตั้งแต่ตำแหน่ง มุมซ้ายบนที่ คอลัมน์ 5 แถว 2 และมุมขวาล่างที่ตำแหน่ง คอลัมน์ 50 แถวที่ 10



2. ให้เขียนโปรแกรมเพื่อสร้างฟังก์ชันคำนวณค่า factorial(n) เพื่อคำนวณหาค่าแฟกทอเรียลของจำนวน n
3. ให้เขียนโปรแกรมเพื่อสร้างฟังก์ชันแบบพารามิเตอร์และมีการ return value ในการคำนวณเลขยกกำลังโดยการส่ง argument เป็นเลขจำนวนฐานและเลขกำลังไปให้แก่ฟังก์ชัน เช่น power(2,4) คือการคำนวณเลขจำนวน 2 ยกกำลัง 4
4. ให้เขียนโปรแกรมเพื่อสร้างฟังก์ชันประเภทมีพารามิเตอร์และมีการ return value ในการคำนวณหาค่าพื้นที่ของรูปเรขาคณิตต่อไปนี้ วงกลม, สามเหลี่ยม และ สี่เหลี่ยม
5. จงเขียนโปรแกรมเพื่อคำนวณการตัดเกรด โดยการเขียนแยกเป็นฟังก์ชัน มีเงื่อนไขการตัดเกรด ดังนี้

   คะแนน 0-49 เกรด F คะแนน 50-59 เกรด D

   คะแนน 60-69 เกรด C คะแนน 70-79 เกรด B

   คะแนน 80-100 เกรด A

   ในโปรแกรมให้สร้างฟังก์ชันต่อไปนี้

   void input() ทำหน้าที่รับคะแนนระหว่างภาคและปลายภาคเรียน

   int summation(int x, int y) ทำหน้าที่ในการรวมคะแนน

   char calculate(int total) ทำหน้าที่ในการตัดเกรด

   void display() ทำหน้าที่ในการแสดงผลข้อมูลของโปรแกรมทั้งหมด
6. ให้เขียนโปรแกรมสร้างฟังก์ชันประเภทมีพารามิเตอร์และมีการ return value เพื่อคำนวณหาผลรวมของเลขอนุกรม 2 ฟังก์ชัน โดยมีชื่อฟังก์ชัน sum_even() และ sum_odd มีการทำงาน ดังตัวอย่าง

   sum_odd(1,10) คือหาผลรวมของ 1+3+5+7+9

   sum_odd(9,20) คือหาผลรวมของ 9+11+13+15+17+19

sum_even(1,10) คือหาผลรวมของ 2+4+6+8

sum_even(9,20) คือหาผลรวมของ 10+12+14+16+18+20

7. Write the function **digitsum(n) ,**which computes and returns the sum of the decimal digits of the integer parameter n.
8. Write the function sort4, which has four parameters. If the integer variables a,b,c and d are available and have been assigned values, we want to write

   sort4(&a, &b,&c,&d);

   to sort four variables, so that, after that call, we have a<=b<=c<=d