EEE102

C++ Programming and Software Engineering II

Lecture 5 Functions

Dr. Rui Lin / Qing Liu

Rui.Lin / Qing.Liu@xjtlu.edu.cn

Room EE512 / EE516

Office hour: 2-4pm, Tuesday & Wednesday

/ Monday & Wednesday

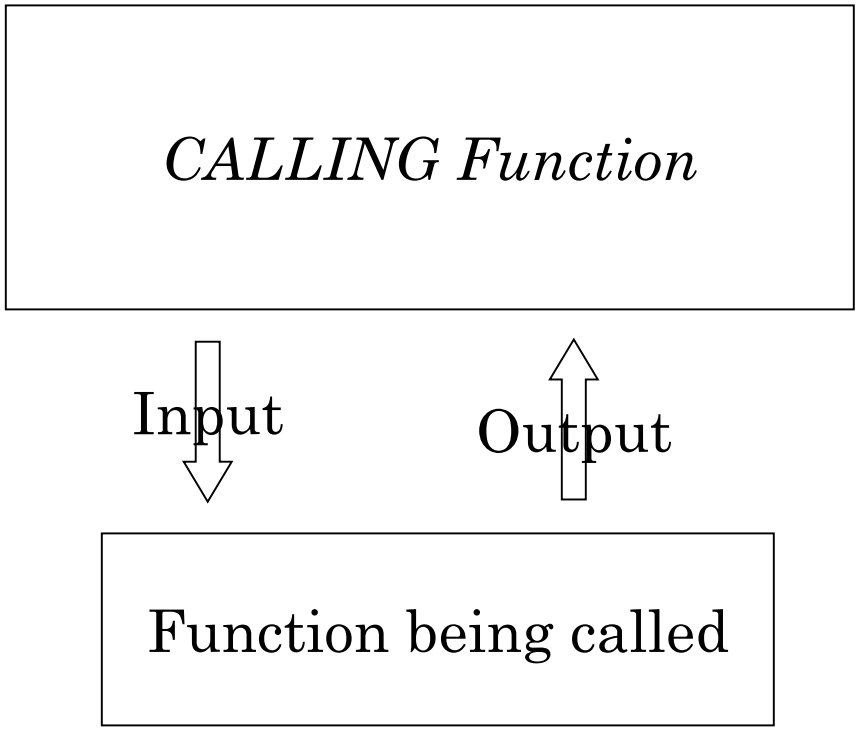
Outline

- Fundamental of functions
  - Types of functions
  - Using functions
  - Information exchanging

- Special use of functions
  - Function overloading
    - Operator overloading
  - Function with default parameters
  - Inline functions

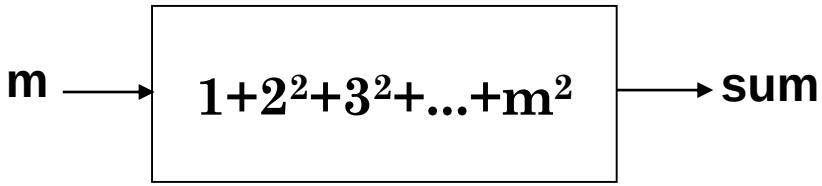
1. Subprogram: functions

- A C++ function is a segment of programme code that performs operations independently.
- It can receive input from and returns results to another function through an interface which is called the *function head*.
- A function should be designed in such a way that it is reusable when required.

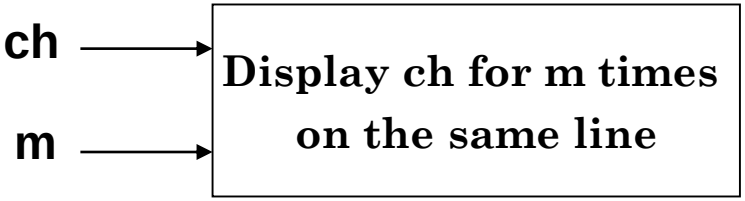


1.1 Types of functions

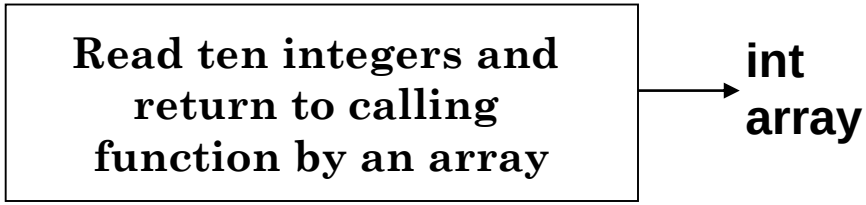
- **Type 1: Functions that take input and return a value**
- **Type 2: Functions that take input but do not return a value**
- **Type 3: Functions do not take input but return a value**
- **Type 4: Functions do not take any input and neither return any value**



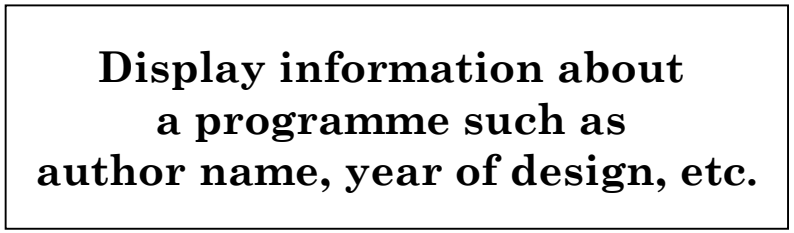
Type 1



Type 2



Type 3



Type 4

Example of Function

- To calculate the sum of a number sequence and display result on screen.

- Problem – similar coding is used and program is lengthy.

mi

```
ni=0;
for (int k=1;k<mi+1;k++)
    ni=ni+k;
```

ni

```
//Calculating the sum of number sequence
#include<iostream>
using namespace std;
int main()
{
    int m1,m2,m3,m4,m5;
    int n1,n2,n3,n4,n5;

    n1=0;m1=5;
    for (int k=1;k<m1+1;k++)
        n1=n1+k;
    cout<<"The sum from 1 to "<<m1<<" is
" <<n1<<endl;

    n2=0;m2=10;
    for (int k=1;k<m2+1;k++)
        n2=n2+k;
    cout<<"The sum from 1 to "<<m2<<" is
" <<n2<<endl;

    //repeating for n3,n4 and n5……
    return 0;
}
```

1.2 Use of functions

- 1. Declaration
  - A function has to be declared before it is used (Prototype): the declaration tells the programme that the name declared is a function.

```
returnType func_name(dataType parameter1, dataType parameter2, …);
```

  - In source code, declaration must be placed before the calling function

- 2. Definition

```
returnType func_name(dataType parameter1, dataType parameter2, …)
{
        Statement1;
        Statement2;
}
```

- 3. Calling

```
func_name(argument1, argument2, …);
```

Using functions (Type 1)

When a function returns a value, it can be used as an expression

```
int sum(int m)
{
    int n=0;
    for (int k=1;k<m+1;k++)
        n=n+k;
    return n;
}
```

```
//Calculating the sum of number sequence.
#include<iostream>
using namespace std;

int sum(int m);

int main(void)
{
    int m1,m2,m3,m4,m5;
    int n1,n2,n3,n4,n5;

    m1=5;
    n1=sum(m1);

    cout<<"The sum from 1 to "<<m1<<" is
" <<n1<<endl;

    m2=10;
    n2=sum(m2);

    cout<<"The sum from 1 to "<<m2<<" is
" <<n2<<endl;

    //repeating for n3,n4 and n5……
    return 0;
}
```

Argument and Parameter

- 1. Argument
  - Variables or constants in the calling function whose values will be passed into a function.

```
func_name(argument1, argument2, …);
```

- 2. Parameter
  - Variables used in a function whose values are passed from the calling function.

```
void main(void)
{
    int n1,m1
    n1=sum(m1);
}

int sum(int m)
{
    int n=0;
    for (int k=1;k<m+1;k++)
        n=n+k;
    return n;
}
```

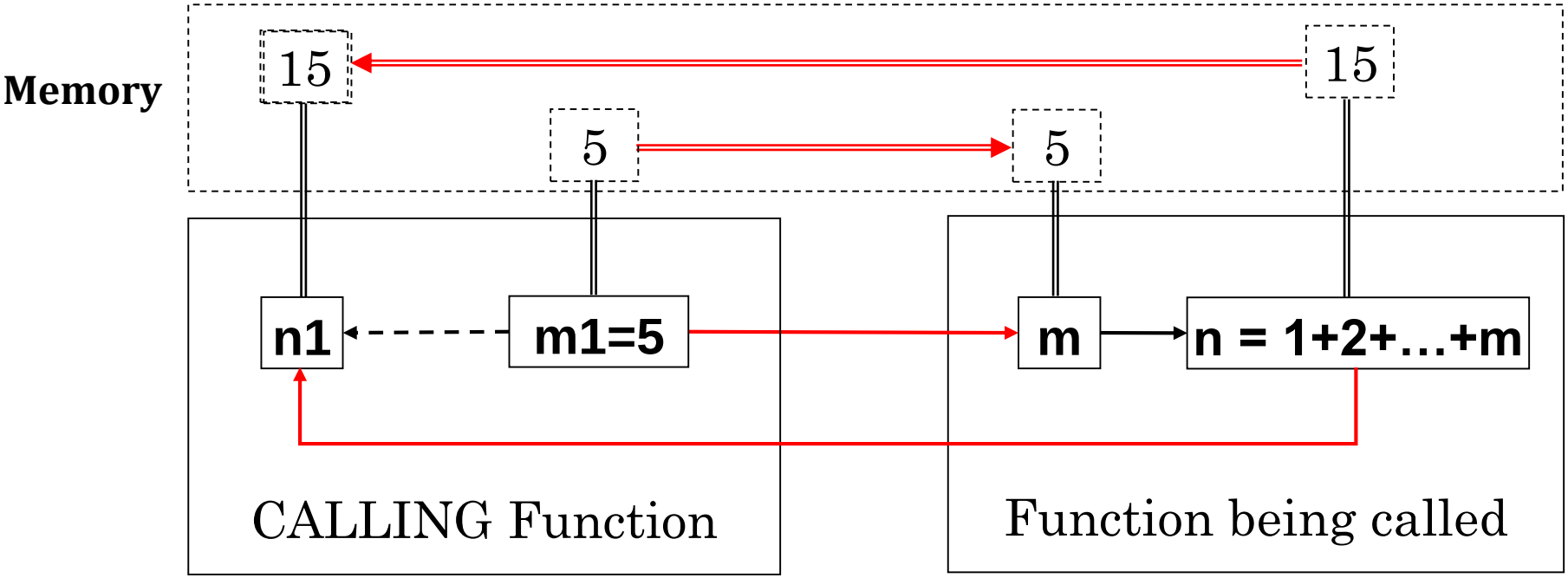
```
returnType func_name(dataType parameter1, dataType parameter2, …)
{
        Statements;
}
```

Using functions (Type 2)

Calling a function which requires input but does not return a value

```
//C++ function, taking input and returning no value
#include <iostream>
#include <cmath>
using namespace std;
void show_sumSquare(float p1, float p2);

int main(void)
{
        float x1,x2;
        cout<<"Type in two real numbers separated by a space "<<endl;
        cin>>x1>>x2;
        show_sumSquare(x1,x2);
        return 0;
}

void show_sumSquare(float p1, float p2)
{
        float value;
        value = pow(p1,2) + pow(p2,2);
        cout<<"The sum of square of "<<p1<<" and "<<p2<<" is "
<<value<<endl;
}
```

1.3.1 Information Exchange – pass by value



- **Pass-by-value**
  - The values of the arguments in the calling function are first copied.
  - The copied values are then passed to the parameters in the called function.
  - Operations in a function **do not affect** the values of the arguments.

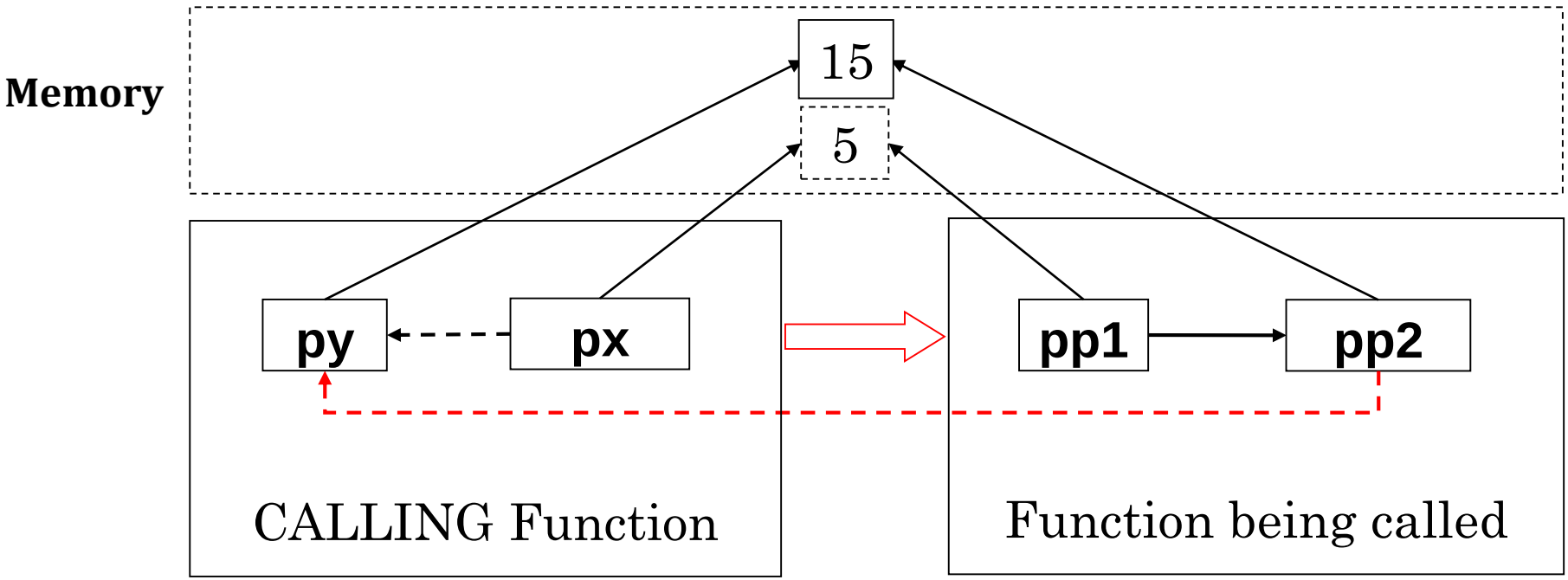
Information Exchange between Two Functions

- **Question:**

  How to return more than one value to the calling function ?

- **Answer:**
  - Pass-by-pointer
  - Pass-by-reference

1.3.3 Information Exchange – pass by pointer



- **Pass-by-pointer**
  - px and pp1 : pointers to same data type
  - py and pp2: pointers to same data type
  - Operations in a function **will affect** the values of the arguments.

Example of pass-by-pointer

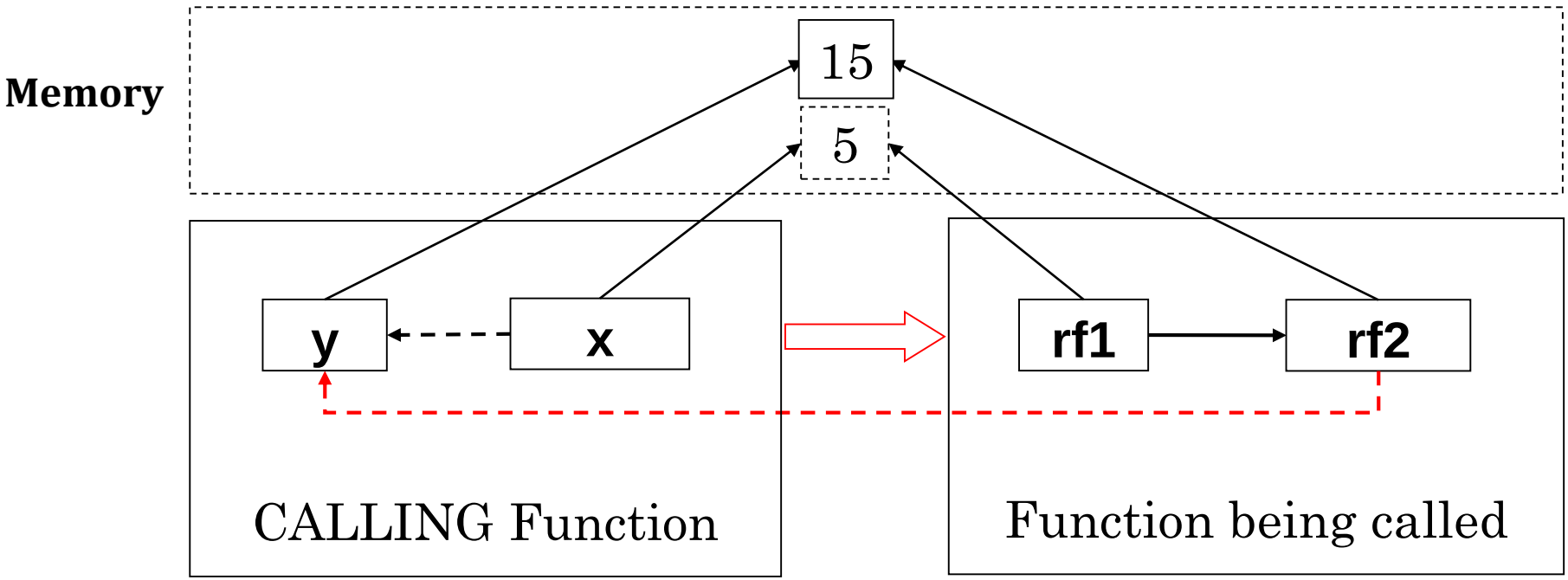
```
//C++ function, calculating the sum of number sequence.
//passing by pointer
#include <iostream>
using namespace std;
void sum(int *pp1, int *pp2);

void main(void)
{
    int px, py, *ptrx, *ptry;
    ptrx=&px;
    ptry=&py;
    cout<< "Please input an integer\n" ;
    cin>>*ptrx;
    sum(ptrx,ptry);
    cout<< "The sum from 1 to " <<px<< " is " <<py<<endl;
}

void sum(int *pp1, int *pp2)
{
    *pp2=0;
    for (int k=1;k<*pp1+1;k++)
        *pp2+=k;
    *pp1=0;
}
```

Watch 1

Name	Value	Type
px	5	int
py	15	int
ptrx	0x0017ff28	int *
	5	int
ptry	0x0017ff1c	int *
	15	int
pp1	0x0017ff28	int *
pp2	0x0017ff1c	int *
k	5	int

Add this statement and try to check the variables values by yourselves.

1.3.3 Information Exchange – pass by reference



- **Pass-by-reference**
  - A reference is another name of a variable
  - x and rf1 are two names of the same variable, i.e. same content in same memory address
  - Operations in a function **will affect** the values of the arguments.

Example of pass-by-reference

```
//C++ function, calculating the sum of number sequence.
//passing by reference
#include <iostream>
using namespace std;
void sum(int &rf1, int &rf2);

void main(void)
{
    int x, y;
    cout<< "Please input an integer\n" ;
    cin>>x;
    sum(x,y);
    cout<< "The sum from 1 to " <<x<< " is " <<y<<endl;
}

void sum(int &rf1, int &rf2)
{
    rf2=0;
    for (int k=1;k<rf1+1;k++)
        rf2+=k;
    rf1=0;
}
```

Watch 1

Name	Value	Type
x	5	int
y	15	int
rf1	5	int &
rf2	15	int &
k	6	int

Add this statement and try to check the variables values by yourselves.

2.1 Function Overloading

- Overloading – using the same function name to create functions that perform different tasks
  - Same function names
  - Different parameter list

- Overloading functions that perform closely related tasks to improve the readability and understandability
  - Do not over use

- Examples:

```
Group 1.
int sum (float a, int b);
int sum (int a, int b);
int sum (int a, int b, int c);
int sum (int a, int *b);
```

VALID!

```
Group 2.
int sum (int a, int b);
float sum (int a, int b);
```

Invalid!

```
Group 3.
int sum (int a, int b);
int sum (int a, int &b);
```

Invalid!

```
Group 4.
int sum (int a, int b);
int sum (int a, int b, int c=0);
```

Invalid!

Example 1: normal function overloading

```
#include<iostream>
using namespace std;

struct complex
{       double real;   double imag;};

complex sum(complex a, double b)
{       a.real+=b;     return a;}

complex sum(complex a, complex b)
{       a.real+=b.real;a.imag+=b.imag;
        return a;}

int main()
{
        double dNum=5.0;
        complex result, cNum={1,2}, cNum2={3,4};
        result=sum(cNum,dNum);
        result=sum(cNum,cNum2);
        cout<<result.real <<"+i" <<result.imag<<endl;
        return 0;
}
```

Example 2: member functions of a class overloading

```
class complex
{
        double real;
        double imag;
public:
        complex(double r=0, double i=0)
        {       real=r; imag=i;}

        complex sum(complex a)
        {       complex temp;
                temp.real=a.real+real;
                temp.imag=a.imag+imag;
                return temp;}

        complex sum(double b)
        {       complex temp;
                temp.real=real+b;
                temp.imag=imag;
                return temp;}

        void display();
};
```

```
int main()
{
        double dNum=5.0;
        complex result1,result2;
        complex cNum(1,2),cNum2(3,4);
        result1=cNum.sum(dNum);
        result2=cNum.sum(cNum2);
        result1.display();
        result2.display();
        return 0;
}
```

```
complex = complex + complex
```

```
complex = complex + real
```

2.2 Operator Overloading

How to calculate the addition, subtraction of two complex numbers?

Add function members to perform the arithmetic calculations

```
class complexClass
{
        double x;
        double y;
public:
        ……
};
void complexClass::assign(complexClass &a)
{       x=a.x;  y=a.y;}
complexClass complexClass::plus(complexClass a)
{       complexClass temp(a.x+x,a.y+y);
        return temp;}
complexClass complexClass::minus(complexClass a)
{       complexClass temp(x-a.x,y-a.y);
        return temp;}
void complexClass::display()
{       cout <<x <<(y>=0?"+":"")<<y <<"i"<<endl;
}
```

Add function members to perform the arithmetic calculations

```
int main()
{
  complexClass a(3,2);
  complexClass b(5,4);
  complexClass result;
  result.assign(a.plus(b));
  result.display();
  result.assign(a.minus(b));
  result.display();
  return 0;
}
```

```
result=a+b;
result=a-b;
```

?

Operator Overloading

- C++ tries to make the user-defined data types behave in much the same way as the built-in types.
  - Eg. C++ permits the addition of two objects with the same syntax for basic types, such as **`object1+object2`**.

- The mechanism of giving such special meanings to an operator is known as ***operator overloading***.
  - We can overload (giving additional meaning to) almost all the C++ operators
  - We cannot change the syntax (operand, precedence, etc.)

Syntax: ***returntype*** **keyword** **operator to be overloaded** **(parameter)**

complexClass **operator** **+** **(complexClass a);**

Add function members to perform the arithmetic calculations

```
void complexClass::operator = (complexClass a)
{
        x=a.x;
        y=a.y;
}
complexClass complexClass::operator +
(complexClass a)
{
        complexClass temp(a.x+x,a.y+y);
        return temp;
}
complexClass complexClass::operator -
(complexClass a)
{
        complexClass temp(x-a.x,y-a.y);
        return temp;
}
```

Use function members

```
int main()
{
        complexClass a(3,2);
        complexClass b(5,4);
        complexClass result;

        result=a+b;
        result.display();

        result=a-b;
        result.display();

        result=a-b-(a+b);
        result.display();

        return 0;
}
```

2.3 Default Arguments

- A default argument is a value that's used automatically if you omit the corresponding actual argument from a function call.
- The default value is set in ***function declaration***.
  - Syntax:

```
// In function declaration
returnType func_name(dataType para1, dataType para2 = value);
```

```
// In function definition
returnType func_name(dataType para1, dataType para2)
{
        statements;
}
```

```
// In calling function
func_name(argu1, argu2); or func_name(argu1);
```

Example 1: normal function with default argument

Function declaration

Function call
omit 2 arguments
omit 1 argument
doesn't omit argument

Function definition

```
#include<iostream>
using namespace std;

double volume(double radius=1, double height=1);

int main()
{
        double a=5, b=2;

        cout<< volume()     <<endl;
        cout<< volume(a)    <<endl;
        cout<< volume(a,b)  <<endl;

        return 0;
}

double volume(double radius, double height)
{
        return 3.14*radius*radius*height;
}
```

Example 1: normal function with default argument

Function declaration

Function call
omit 2 arguments
omit 1 argument
doesn't omit argument

Function definition

```
#include<iostream>
using namespace std;

double volume(double radius=1, double height=1);

int main()
{
        double a=5, b=2;

        cout<<  volume(1,1) <<endl;
        cout<<  volume(5,1) <<endl;
        cout<<  volume(5,2) <<endl;

        return 0;
}

double volume(double radius, double height)
{
        return 3.14*radius*radius*height;
}
```

Example 2: method of a class with default argument

- Constructor of the class complexClass

```
class complexClass
{
        double x;
        double y;
public:
        complexClass(){}
        complexClass(double r, double i=0);
};
complexClass::complexClass(double r, double i)
{
        x=r;
        y=i;
}
```

```
int main()
{
    complexClass c0;
    complexClass c1(-9);
    complexClass c2(-9,5);
    return 0;
}
```

Rules for using Default Arguments

- When you use a function with an argument list, you must add default values from right to left.

```
int exch1(int n=1, int m=2, int k=3);          // VALID
int exch2(int n, int m=2, int k=3);            // VALID
int exch3(int n, int m, int k=3);              // VALID
int exch4(int n, int m=2, int k);              // INVALID
int exch5(int n=1, int m=2, int k);            // INVALID
int exch6(int n=1, int m, int k);              // INVALID
```

- The actual arguments are assigned to the corresponding formal arguments from left to right. You cannot skip over arguments.

```
temp = exch1(10,20,30);    // VALID
temp = exch1(10,  ,30);    // INVALID, cannot skip m
temp = exch1(10,20);       // VALID, equivalent to exch1(10,20,3)
temp = exch2(10);          // VALID, equivalent to exch2(10,2,3)
temp = exch3(10);          // INVALID, no value for m
```

2.4 Inline function

- An inline function is a function that expanded in line when it is invoked. – the compiler replaces the function call with the corresponding function code.
  - To reduce the cost of calls to small functions.
  - Every time a function is called, it takes a lot of extra time in executing a series of instructions for tasks such as jumping to the function, saving registers, pushing arguments into the stacks, and returning to the calling function.
- Syntax:

```
// In function declaration
inline returnType func_name( parameter list );
```

```
//Calculating the sum of number
sequence
#include<iostream>
using namespace std;
int main()
{
    int m1,m2,m3,m4,m5;
    int n1,n2,n3,n4,n5;

    m1=5;
    n1=sum(m1);
    m2=10;
    n2=sum(m2);

    //repeating for n3,n4 and n5……
    return 0;
}
```

```
inline int sum(int m)
{   int n=0;
    for (int k=1;k<m+1;k++)
        n=n+k;
    cout<<"The sum to "<<m<<" is ";
    cout<<n<<endl;
    return n;}
```

A regular function transfers program execution to a separate function.

```
//Calculating the sum of number
sequence
#include<iostream>
using namespace std;
int main()
{
    int m1,m2,m3,m4,m5;
    int n1,n2,n3,n4,n5;
    m1=5;
    n1=0;
    for (int k=1;k<m1+1;k++)
      n1=n1+k;
    cout<<"The sum to "<<m1<<" is ";
    cout<<n1<<endl;
    m2=10;
    n2=0;
    for (int k=1;k<m1+1;k++)
      n2=n2+k;
    cout<<"The sum to "<<m2<<" is ";
    cout<<n2<<endl;

    //repeating for n3,n4 and n5……
    return 0;
}
```

An inline function replaces the function call with inline code.