AMA2222 Principles of Programming

Leung Man Kin, Adam
Instructor

adam.leung@polyu.edu.hk
TU720

Chapter 4: Function
defining a functions, return value types,
function prototype, parameters with default values,
overloading a function, global and local variable,
pass-by-reference, lambda function

Function

Functions can be used to define reusable code. The function definition must be placed before all function calls. A function definition consists of its function name, parameters, return value type, and body. Notice that the execution of **`return`** statement will exit the function.

In constructing a function, always be clear in your mind: what are the input, process, and output of your function?

Syntax for function:

```
returnValueType functionName(dataType1 parameter1,…)
{
    statement(s);              // function body
    return returnValue;
}
```

Example Program 4.1

If we are asked to add up integers from 1 to 10, we can simply write a program with for loop. What if we are also asked to add up from 13 to 25 and 78 to 87? In general, we can write a function that add up integers between any two numbers.

What are the inputs?	Two numbers.
What is the process?	Add up the numbers in between.
What is the output?	A number, the sum.

```
int sum(int i1, int i2)
{
    int s = 0;
    for (int i = i1; i <= i2; i++)
        s += i;
    return s;
}
```

```
1  // 4.1a Defining a function
2  #include <iostream>
3  using namespace std;
4  int sum(int i1, int i2)
5  {
6          int s = 0;
7          for (int i = i1; i <= i2; i++)
8                  s += i;
9          return s;
10 }
11
12 int main()
13 {
14         cout << "Sum from 1 to 10 is " << sum(1,10) << endl;
15         cout << "Sum from 13 to 25 is " << sum(13,25) << endl;
16         cout << "Sum from 78 to 87 is " << sum(78,87) << endl;
17         return 0;
18 }
```

Input two numbers i1<i2, add up all integers in between inclusively, output the sum.

The main body of this program.
Calls sum to do the calculation

main: hello **sum**, what is the answer of adding from **1** to **10**?

input ➡

sum: that would be **55**.

⬅ **return value**

main: thanks, then I will print "The sum from 1 to 10 is 55".

Classwork exercise 4.1

Write a function called `mag` that calculates the magnitude of a vector in 3D space, i.e. $\|\vec{v}\| = \sqrt{x^2 + y^2 + z^2}$.

Write a program that prompts user to enter the three components of a vector, then output the magnitude of this vector.

```
Please enter the three components of the vector: 1 2 3
The magnitude of this vector is 3.74166
```

```cpp
// CW4.1 Vector magnitude
#include <iostream>
#include <cmath>
using namespace std;
double mag(double x, double y, double z)
{
        double m = 0;
        m = sqrt(x*x + y*y + z*z);
        return m;
}
int main()
{
        cout << "Please enter the three components of the vector: ";
        double a, b, c;
        cin >> a >> b >> c;
        cout << "The magnitute of this vector is " << mag(a,b,c);
        return 0;
}
```

Void function

A function does not necessarily return a value. For such function, we can use `void` instead of any other variable data types.

Even without return value, a void function can still perform tasks like using cout to print.

Syntax for void function:

```
void functionName(list of parameters)
{
    statement(s);              // function body
}
```

```
1  // 4.1b Defining a function
2  #include <iostream>
3  using namespace std;
4  int sum(int i1, int i2)
5  {
6          int s = 0;
7          for (int i = i1; i <= i2; i++)
8                  s += i;
9          return s;
10 }
11 void printSum(int i1, int i2)
12 {
13         cout << "Sum from " << i1 << " to " << i2
14                 << " is " << sum(i1,i2) << endl;
15 }
16 int main()
17 {
18         printSum(1,10);
19         printSum(13,25);
20         printSum(78,87);
21         return 0;
22 }
```

Input two numbers i1<i2, add up all integers in between inclusively, output the sum.

Print the result of adding up all integers in between two numbers.

The main body of this program. Calls printSum, which further calls sum.

main: hello **printSum**, please print the sum of adding from 1 to 10. ➡
printSum: but I dunno the answer, let me call **sum** to ask.

printSum: hi **sum**, do you know the answer of adding from 1 to 10? ➡
sum: sure, the answer is 55.
⬅

printSum: hi **main**, I am going to print "The sum from 1 to 10 is 55".
main: thanks a lot.

Classwork exercise 4.2

Write a void function called `love` that reads an positive integer and print the "I love you!" for that many times.

Write a program that prompts user to enter the a positive integer n and put it into the love function.

```
Please enter a number: 3
I love you!
I love you!
I love you!
```

```cpp
// CW4.2 love
#include <iostream>
#include <cmath>
using namespace std;
void love(int num)
{
	for (int i = 1; i <= num; i++)
		cout << "I love you!" << endl;
}
int main()
{
	int n;
	cout << "Please enter a number: ";
	cin >> n;
	love(n);
	return 0;
}
```

Boolean function

A **boolean function** returns a boolean value which is either **true** or **false**. This output value can be used as a logical condition.

Example program 4.3

Write a function called `isprime` that checks if the input value is a prime number.

Write a program that outputs all prime numbers within a given number n.

```
The prime numbers within 20:
2 3 5 7 11 13 17 19
```

Steps of thinking:

1) What is the output type of the function
2) What is/are the input/inputs of the function
3) What is the criteria of being a prime number
4) Given the function, how to list out all the prime numbers

1) boolean
2) an integer
3) not divisible by numbers rather than 1 and itself
4) use a for-loop to go over all numbers from 2 to n,
   then use if-statement with the function being the condition

```cpp
// 4.2 Boolean function
#include <iostream>
using namespace std;
bool isprime(int num)
{
	bool flag = true;
	for (int i = 2; i < num; i++)
		if (num % i == 0) flag = false;
	return flag;
}
int main()
{
	const int n = 20;
	cout << "The prime numbers within " << n << " :" << endl;
	for (int i = 2; i <= n; i++)
		if (isprime(i))
			cout << i << " ";
	return 0;
}
```

```
 4 bool isprime(int num)
 5 {
 6           bool flag = true;
 7           for (int i = 2; i < num; i++)
 8                     if (num % i == 0) flag = false;
 9           return flag;
10 }
```

prime(6)
flag = true
i = 2, flag = false
i = 3, flag = false
i = 4
i = 5
return flag (= false)

For a compound number, when i equal its factors (rather than 1 and itself) the flag is turned to false.

prime(5)
flag = true
i = 2
i = 3
i = 4
return flag (= true)

For a prime number, i going from 2 to num-1 cannot be its factor. So flag remains true.

Function prototype

A **function prototype** declares a function without having to implement it. It is a function header without implementation. The implementation is given later in the program.

Before a function is called, its header must be declared. One way to ensure this is to place the definition before all function calls. Another approach is to define a function prototype before the function is called.

If a program involve several functions that might call another, it is hard to decide which function to be implemented first. It is better to declare all function prototype in the beginning of the program.

```
1  // 4.1c Defining a function
2  #include <iostream>
3  using namespace std;
4  int sum(int i1, int i2);
5  int main()
6  {
7           cout << "Sum from 1 to 10 is " << sum(1,10) << endl;
8           cout << "Sum from 13 to 25 is " << sum(13,25) << endl;
9           cout << "Sum from 78 to 87 is " << sum(78,87) << endl;
10          return 0;
11 }
12 int sum(int i1, int i2)
13 {
14          int s = 0;
15          for (int i = i1; i <= i2; i++)
16                    s += i;
17          return s;
18 }
```

function prototype

function implementation can be put in a later part

Function parameters with default values

We can also set up default value for some input parameters. Without inputting a value, those parameter will be set to its default value. To assign a default value to a function parameter, simply use the following syntax within the function input:

```
(dataType1 parameter1 = defaultValue1,…)
```

Notice that we must put parameters without default values before those with default values in the function inputs.

When the function is called, input values will be fitted to the parameters **in front** of the function. Those parameters without input value will be assigned with the default value.

Example 4.1d

```
1  // 4.1d Defining a function
2  #include <iostream>
3  using namespace std;
4  int sum(int i1=1, int i2=10)
5  {
6           int s = 0;
7           for (int i = i1; i <= i2; i++)
8                    s += i;
9           return s;
10 }
11
12 int main()
13 {
14          cout << "Sum from 1 to 10 is " << sum() << endl;
15          cout << "Sum from 5 to 10 is " << sum(5) << endl;
16          cout << "Sum from 78 to 87 is " << sum(78,87) << endl;
17          return 0;
18 }
```

set up default values for the two parameters

the function can accept 0, 1 or 2 input values, the remaining are filled up by default values

```
sum()          // i1=1, i2=10
sum(5)         // i1=5, i2=10
sum(78,87)     // i1=78, i2=87
```

Classwork 4.3 This function below calculates dot product of two two-dimensional vectors u=(u1,u2) and v=(v1,v2). By default, the second vector is (1,1).

```
int dp(int u1, int u2, int v1=1, int v2=1){
   return (u1*v1 + u2*v2);}
```

Based on the function below, what will be the results of the following?

```
dp(3,4)      u1=3, u2=4, v1=1(default), v2=1(default)

dp(3,4,2,5) u1=3, u2=4, v1=2, v2=5

dp(3,4,2)    u1=3, u2=4, v1=2, v2=1(default)
```

Notice that in our input to the function, those parameters are read in the same order of how the function was defined.

```
dp(3,4)      u1=3, u2=4, v1=1(default), v2=1(default)

dp(3,4,2,5) u1=3, u2=4, v1=2, v2=5

dp(3,4,2)    u1=3, u2=4, v1=2, v2=1(default)
```

Hence they give the following result:

```
dp(3,4)      3*1+4*1=7

dp(3,4,2,5) 3*2+4*5=26

dp(3,4,2)    3*2+4*1=10
```

Overloading a function

Overloading functions enables you to define the functions with the same name as long as their parameter list is different.

In the example below,
`maximum(n1,n2)` is a function that returns the greatest value of the two inputs n1 and n2.
`maximum(n1,n2,n3)` is a function that returns the greatest value of the three inputs n1, n2 and n3.

These two functions have the same name but different numbers of inputs. We say the function `maximum` is overloaded but they can still be distinguished in C++.

Example program 4.3

```
1  //4.3 Overloading functions
2  #include <iostream>
3  using namespace std;
4  double maximum(double n1, double n2)
5  {
6           if (n1>=n2)
7                    return n1;
8           else
9                    return n2;
10 }
11 double maximum(double n1, double n2, double n3)
12 {
13          if (n1>=n2 && n1>=n3)
14                   return n1;
15          if (n2>=n1 && n2>=n3)
16                   return n2;
17          if (n3>=n1 && n3>=n2)
18                   return n3;
19 }
20 int main()
21 {
22          cout << "The maximum between 3 and 4 is " << maximum(3,4) << endl;
23          cout << "The maximum among 3, 4, and 5 is " << maximum(3,4,5) << endl;
24          return 0;
25 }
```

Global and local variables

scope: the scope of a variable is the part of the program where the variable can be referenced. A variable declared in the initialization part of a for statement has its scope in the entire loop. A variable declared inside a for loop body has its scope limited in the loop body from its declaration to the end of the block that contains the variable.

local variable: variable defined inside a function/loop, will be destroyed after the function/loop is executed.

global variable: variable declared outside all functions, are accessible to all functions in their scope, default to be zero.

If a function has a local variable with the same name as a global variable, only the local variable can be seen from the function.

Example program 4.4 a, b

What will be the output value of x?
Both programs can be executed, but one of them got a problem……

```
#include <iostream>
using namespace std;
int x = 2;
int main()
{
        for (int i = 1; i <= 10; i++)
        {
                x += 1;
        }
        cout << x << endl;
        return 0;
}
```

```
#include <iostream>
using namespace std;
int x = 2;
int main()
{
        for (int x = 1; x <= 10; x++)
        {
                x += 1;
        }
        cout << x << endl;
        return 0;
}
```

Example program 4.4a

```
#include <iostream>
using namespace std;
int x = 2;
int main()
{
        for (int i = 1; i <= 10; i++)
        {
                x += 1;
        }
        cout << x << endl;
        return 0;
}
```

x: global variable
scope of x: the whole program

i: local variable
scope of i: only in this for-loop

x has an initial value of 2.
In each iteration of the for-loop, x is increased by 1.
After 10 iterations, x is increased to 12.
i is used only locally and it is destroyed after the for-loop.

Example program 4.4b (A fail demonstration!)

```
#include <iostream>
using namespace std;
int x = 2;
int main()
{
        for (int x = 1; x <= 10; x++)
        {
                x += 1;
        }
        cout << x << endl;
        return 0;
}
```

x: global variable
x: local variable

There is a conflict that the same name x is used for both global and local variable. Bad habit.

x has an initial value of 2.
In the for-loop, x is used locally as a counter. For each iteration, x is increased by 1 in the loop body, and then increased by 1 in the x++.
Locally, x is increased up to 10.
However, this local variable is destroyed after the for-loop is done.
The final output of x is just 2, the initial value.

Statics local variable

We have learnt that a local variable will be destroyed after the function/loop is executed. What if we want to keep this value so that it can be accessed in the other parts of the program?

static local variable: variable that is defined locally, but permanently allocated in the memory of the program

To declare a static local variable, just put the keyword `static` before the data type and variable name. It can only be initiated in the first call.

We will see the different between statics local variable and local variable in the following example.

Example program 4.5

In this example, evaluate the output. Can you tell what type of variables are x, y, and z?

```
1  //4.5 statics local variable
2  #include <iostream>
3  using namespace std;
4  int z = 1;
5  void f1()
6  {
7            static int x = 1;
8            int y = 1;
9            x++;
10           y++;
11           z++;
12           cout << "x is " << x << endl;
13           cout << "y is " << y << endl;
14           cout << "z is " << z << endl;
15 }
16 int main()
17 {
18           f1();
19           f1();
20           return 0;
21 }
```

```
x is 2
y is 2
z is 2
x is 3
y is 2
z is 3
```

x: static local variable

y: local variable

z: global variable

int z = 1

```
f1()
      static int x = 1;
      int y = 1;
      x++; (x=2)
      y++; (y=2)
      z++; (z=2)
      y destroyed
```

output:

x is 2
y is 2
z is 2

```
f1()
      static int x = 1; (ignored)
      int y = 1;
      x++; (x=3)
      y++; (y=2)
      z++; (z=3)
      y destroyed
```

output:

x is 3
y is 2
z is 3

Passing arguments by reference

When the value of a variable is passed to the parameter of a function, this is referred to as **pass-by-value**. The variable is not affected, regardless of the changes made to the parameter inside the function.

To retain the changes made by a function, we need a reference variable which is an alias of another variable. This is known as **pass-by-reference**. To declare a reference variable, place the ampersand (&) in front of the variable.

Example program 4.6
Compare the output of the following programs

```
1  //4.6a pass-by-value
2  #include <iostream>
3  using namespace std;
4  void swap(int x, int y)
5  {
6            int temp = x;
7            x = y;
8            y = temp;
9  }
10 int main()
11 {
12           int num1 = 1;
13           int num2 = 2;
14           cout << num1 << "," << num2 << endl;
15           swap(num1,num2);
16           cout << num1 << "," << num2 << endl;
17           return 0;
18 }
```

```
1,2
1,2
```

The values of num1 and num2, which are 1 and 2 respectively, are input to the function and stored as the values of x and y inside.

Any changes of the values of x and y within the function would not affect num1 and num2.

Example program 4.6
Compare the output of the following programs

```
1  //4.6b pass-by-reference
2  #include <iostream>
3  using namespace std;
4  void swap(int &x, int &y)
5  {
6            int temp = x;
7            x = y;
8            y = temp;
9  }
10 int main()
11 {
12           int num1 = 1;
13           int num2 = 2;
14           cout << num1 << "," << num2 << endl;
15           swap(num1,num2);
16           cout << num1 << "," << num2 << endl;
17           return 0;
18 }
```

```
1,2
2,1
```

num1 and num2 are input as reference variables to the function as x and y, labelled by "&" in front.

Any changes of the values of x and y within the function would also affect the values of num1 and num2.

Classwork exercise 4.4

A student wrote the following function to find the minimum and maximum between two values a and b. What is wrong in the program?

```
 1 #include <iostream>
 2 using namespace std;
 3 void minMax(double n1, double n2, double min, double max)
 4 {
 5           if(n1 < n2)
 6           {
 7                     min = n1;
 8                     max = n2;
 9           }
10           else
11           {
12                     min = n2;
13                     max = n1;
14           }
15 }
16 int main()
17 {
18           double a = 5, b = 6, x, y;
19           minMax(a, b, x, y);
20           cout << "min is " << x << " and max is " << y << endl;
21           return 0;
22 }
```

These two numbers are to be updated. Use pass by reference.

```
 1 #include <iostream>
 2 using namespace std;
 3 void minMax(double n1, double n2, double &min, double &max)
 4 {
 5           if(n1 < n2)
 6           {
 7                     min = n1;
 8                     max = n2;
 9           }
10           else
11           {
12                     min = n2;
13                     max = n1;
14           }
15 }
16 int main()
17 {
18           double a = 5, b = 6, x, y;
19           minMax(a, b, x, y);
20           cout << "min is " << x << " and max is " << y << endl;
21           return 0;
22 }
```

Lambda function

In the beginning of this chapter, we have learnt that a return data type of a function must be declared in the very beginning. Starting from C++14, a new feature is called **automatic return type**. Using the keyword **`auto`** to substitute the data type of the function, a suitable data type based on the return value will be allocated automatically.

You may need to change the setting in your development platform in order to compile with C++14 by the following steps:

Tools -> Compiler Options -> General
-> Add the following commands when calling the compiler:
`-std=c++14`

Another feature of using **auto** is to define a function inline, which means within the main part of program. Such function is also called **lambda function**. It can include existing variables in the program in addition to input parameters. The syntax to define such a function inline is as follows:

```
auto functionName = [variable1, …](datatype parameter1, …)
{
    statement(s);
    return returnValue;
};
```

Example program 4.7 Consider a trading company which help customers to buy products in Japan and ship to Hong Kong. However, the exchange rate and markup are two fluctuating factors. Write a program that allows the customer to key in the original price (in YEN) and convert it to the selling price (in HKD).

```cpp
//4.7 lambda function
#include <iostream>
using namespace std;
int main()
{
        double rate = 0.06;
        double markup = 0.3;
        double jprice, hkprice;
        auto convert = [rate, markup](double x)
        {
                return x*rate*(1+markup);
        };
        cout << "Enter the original price in yen: ";
        cin >> jprice;
        hkprice = convert(jprice);
        cout << "Our selling price in hkd is " << hkprice << endl;
        return 0;
}
```

variables defined within the program, which can be included in the lambda function

input parameter to the lambda function

calculate the selling price using the convert function, with jprice as input