C++

FOR BEGINNERS

A BEGINNER'S GUIDE

Ray Yao

C + +

In 8 Hours

By Ray Yao

For Beginners

Ray Yao

About the Author

Ray Yao:

Certified PHP engineer by Zend, USA

Certified JAVA programmer by Sun, USA

Certified SCWCD developer by Oracle, USA

Certified A+ professional by CompTIA, USA

Certified ASP. NET expert by Microsoft, USA

Certified MCP professional by Microsoft, USA

Certified TECHNOLOGY specialist by Microsoft, USA

Certified NETWORK+ professional by CompTIA, USA

Ray Yao’s books:

Linux Command Line (eBook)

Linux Command Line (Paper Book)

JAVA 100 Tests, Answers & Explanations

PHP 100 Tests, Answers & Explanations

JavaScript 100 Tests, Answers & Explanations

JAVA in 8 Hours

PHP in 8 Hours

JavaScript in 8 Hours

C++ in 8 Hours

AngularJS in 8 Hours

JQuery in 8 Hours

JavaScript 50 Useful Programs

Preface

C++ in 8 Hours is a useful book for beginners. You can learn complete primary knowledge of C++ fast and easily. The straightforward definitions, the plain and simple examples, the elaborate explanations and the neat and beautiful layout feature this helpful and educative book. You will be impressed by the new distinctive composing style. Reading this book is a great enjoyment! You can master all essential C++ skill in quickly.

Source Code for Download

This book provides source code for download; you can download the source code for better study, or copy the source code to your favorite editor to test the programs. The download link of the source code is at the last page of this book.

Start coding today!

Table of Contents

Source Code for Download

Hour 1

Start C++

Install C++

Many C++ compilers and editors can be used to run C++ programs; you can choose one of them to write, edit, compile and execute your C++ codes.

Online C++ Compiler & Editor

You don’t have to install C++ compiler & editor to your local computer, an online C++ compiler & editor is available in a following website:

http://www.tutorialspoint.com/codingground.htm

When running the online C++ compiler & editor, you can type C++ codes into the editor, click “Compile”, then click “Execute”, the result will appear.

Dev-C++ is an excellent C++ Compiler & Editor

Maybe you want to install free “Dev-C++” to your local computer, please download it by the following website.

http://www.bloodshed.net/dev/devcpp.html.

If you don’t want “Dev-C++”, please download and install “Visual C++ Express”.

Download Visual Studio Express

Visual Studio Express editions provide free tools to develop applications for a specific platform, such as **Visual C++**, Visual C#, Visual Basic, Asp.net and Windows applications.

Visual C++ contains all C++ libraries, C++ tools building and running C++ programs. Visual C++ is an IDE that can create a C++ development environment.

To download a free edition of Visual Studio Express, go to

https://www.visualstudio.com

Install Visual Studio Express

In Visual Studio Express, Visual C++ is not installed by default. When installing, be sure to choose Custom installation and then choose the Visual C++ components you require.

What is C + +?

C++ is a general-purpose programming language, which is an extension of C language. C++ is an object-oriented programming language that is used extensively.

Example 1.1

```
#include <iostream>
using namespace std;
int main() {
cout << “Hello World!” << endl;
return 0;
}        (Output:   Hello World!)
```

Explanation:

“#include <iostream>” means including an input/output library named **iostream** that helps input or output operation.

“using namespace std” means that “std” is a standard namespace, which can solve the name conflict problems.

“int” is a return type, it specify the return type of a function.

“main( )” is an obligatory function that runs main code of C++.

“cout <<” displays or show contents.

“return 0” indicates the program run successfully.

Run First Programming

Write C++ codes to your favorite editor.

```
#include <iostream>
using namespace std;
int main() {
        cout << “Hello World!” << endl;
        return 0;
}
```

Save, Compile and Run the Program

Output:

Hello World!

Congratulation! The first C++ program runs successfully!

C++ Comments

```
//


/*  ……  */
```

C++ comments is used to explain the current code, describe what this code is doing. C++ compiler will ignore comments.

// is used to single line comment.

/*…*/ is used to multi line comments

Example 1.3

```
cout << “Hello World” ;   // show “Hello World”
/*  cout << is a C++ output command, meaning display or print.
*/
```

Explanation:

// show “Hello World” is a single line comment.

/* cout << is a C++ output command, meaning display or print. */ is a multi line comment.

Note: Each C++ command ends with semicolon”**;**”.

Output Commands

```
cout << “   ”;  // print the result at the same line.


cout << “   ” << endl;  // print the result at the
different line.
```

Example 1.4

cout << ” 1 “;

cout << ” 2 “;

cout << ” 3 ”;

(Output: 1 2 3)

cout << ” 1 ” << **endl** ;

cout << ” 2 ” << **endl** ;

cout << ” 3 ” << **endl** ;

(Output:)

1

2

3

Explanation:

cout << “ ”; // print the result at the same line.

cout << “ ” << endl; // print the result at the different line.

Escaping Characters

The “ \ ” backslash character can be used to escape characters.

\n outputs content to the next new line.
\r makes a return
\t makes a tab
\’ outputs a single quotation mark.
\” outputs a double quotation mark.

Example 1.5

```
#include <iostream>
using namespace std;
int main( ) {
cout << “  \” Hello World! \”  “;
return 0;
}
```

(Output: **“**Hello World!**”**)

Explanation:

\” outputs a double quotation mark. Note that “Hello World” has a double quotation with it. Another sample:

cout << “Hello **\t\t\t** World” ; // Note it has three taps. (Output: Hello World)

C++ Keywords

Some words are only used by C++ language itself. They may not be used when choosing identifier names for variable, function, and labels. The following words are C++ keywords:

asm	auto	break
delete	case	catch
char	class	const
continue	default	do
double	else	enum
float	friend	for
goto	if	inline
extern	short	int
long	new	operator
register	private	protected
public	return	signed
static	struct	switch
this	throw(s)	typedef
union	unsigned	virtual
try	void	while

Example 1.6

const // *const* is a keyword of C++

continue // *continue* is a keyword of C++

Explanation:

Above words, “const” and “continue” is C++ reserved words, which may be not used as variable name, function name, and label name.

Data Types

C++ five basic data types are listed in the following:

Data Types	Explanation
char	a character
string	several characters
int	an integer number
float	a floating point number with 6 decimals
double	a floating point number with 10 decimals
bool	a value with true or false

Example 1.7

“hello”

168

0.123456

0.0123456789

true

Explanation:

The data type of “hello” is a string.

The data type of 168 is an int

The data type of 0.123456 is a float

The data type of true is a bool.

Create a Variable

Variable is a symbolic name associated with a value.

Using following syntax can create a variable and value.

```
dataType variableName = value;
```

Example 1.8

```
void main (  ) {
char  mychar = ‘m’;
int myinteger = 168168;
float myflt = 28.98;
double mydbl = 0.0123456789
boolean mybool = true;
}
```

Explanation:

mychar is a name of variable, its value is “m”.

myinteger is a name of variable, its value is 168168.

mybool is a name of variable, its value is true.

“main ( ) { }” declares a main method.

Arithmetical Operators

Operators	Running
+	add, connect strings
-	subtract
*	multiply
/	divide
%	get modulus
++	increase 1
- -	decrease 1

% modulus operator divides the first number by the second number and returns the remainder. e.g. 9%2=1.

Example 1.9

```
#include <iostream>
using namespace std;
int main( ) {
int a = 200, b = 100;
int sum  = a + b;  cout<< sum<<endl;
int divi = a / b;  cout<< divi<<endl;
int modu = a % b;  cout<< modu<<endl;
cout << ++a << endl;
return 0;
}
```

Output:

```
300
2
0
201
```

Explanation:

“int sum = a + b” means 100 plus 200.

“int divi = a / b” means 200 divide 100.

“int modu = a % b” means 200 modulus 100.

“++a” increases its value by 1.

Logical Operators

Operators	Equivalent
&&	and
||	or
!	not

After using logical operators, the result will be true or false.

“1” represents true, “0” represents false.

Example 1.10

```
bool x=true; bool y=false;
bool a= x && y;   cout << a << endl;   // output: 0
bool b=x || y;   cout << b << endl;   // output: 1
bool c=! x;   cout << c << endl;   // output:  0
```

Explanation:

“1” represents true, “0” represents false.

true && true; returns true;	true && false; returns false;	false &&false; returns false;
true II true; returns true;	true II false; returns true;	false II false; return false;
! false; returns true;	**!** true; returns false;	

Assignment Operators

Operator	Example	Explanation
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Example 1.11

```
#include <iostream>
using namespace std;
int main( ) {
int x=200; int y=100;
cout <<“200+=100 equals ” << (x+=y)<< endl;
cout <<“300-=100 equals ” << (x-=y)<< endl;
cout <<“200*=100 equals ” << (x*=y)<< endl;
cout <<“20000%=100 equals “<< (x%=y)<< endl;
return 0;
}
```

Explanation:

x+=y; // x=x+y, x=200+100, output 300.

x-=y; // x=x-y, x=300-100, output 200.

x*=y; // x=x*y, x=200*100, output 20000.

x%y; // x=%y, x=2000000%100, output 0.

Comparison Operators

Operators	Running
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal
!=	not equal

After using comparison operators, the result will be true or false.

Example 1.12

```
#include <iostream>
using namespace std;
int main( ) {
int x=200; int y=100;
bool result1 = (x>y); cout << result1<< endl;
bool result2 = (x<=y); cout << result2<< endl;
bool result3 = (x!=y ); cout << result3 << endl;
return 0;
};
```

Output:

1

0

1

Explanation:

“1” represents true.

“0” represents false.

x>y // test 200>100, output true.

x<=y // test 200<=100, output false.

x**!**=y // test 200**!**=100, output true.

Exercises

Calculation

Write C++ codes to your favorite editor.

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

int main(){
int a=10, b=20, c=30;
int sum;
sum = a + b * c;
cout << “a+b*c = “<< sum << endl;
return 0;
}
```

Save, Compile and Run the Program.

Output:

a+b*c = 610

Hour 2

Statements

If Statement

```
if ( test-expression ) {    // if true do this;    }
```

“if statement” executes codes inside { … } only if a specified condition is true, does not execute any codes inside {…} if the condition is false.

Example 2.1

```
#include <iostream>
using namespace std;
int main( ) {
int a=200;
int b=100;
if (a>b){
cout << “a is greater than b.”;
}
return 0;
}    (Output:   a is greater than b.)
```

Explanation:

(a>b) is a test expression, namely (200>100), if returns true, it will execute the codes inside the { }, if returns false, it will not execute the codes inside the { }.

If-else Statement

```
if ( test-expression) {   // if true do this;   }
else  {  // if false do this;   }
```

“if…else statement” runs some code if a condition is true and another code if the condition is false

Example 2.2

```
#include <iostream>
using namespace std;
int main( ) {
int a=100; int b=200;
if (a>b) {
cout << “a is greater than b.”;
}
else {
cout << “a is less than b”;
}
return 0;
}       (Output:  a is less than b.)
```

Explanation:

(a>b) is a test expression, namely (100>200), if returns true, it will output ”a is greater than b.” if returns false, it will output “a is less than b”.

Switch Statement

```
switch ( variable )
{ case 1: if equals this case, do this;  break;
  case 2: if equals this case, do this;  break;
  case 3: if equals this case, do this;  break;
  default : if not equals any case, run default code;
break;
}
```

The value of variable will compare each case first, if equals one of the “case” value; it will execute that “case” code. “break;” terminates the code running.

Example 2.3

```
#include <iostream>
using namespace std;
int main( ) {
int number=20;
switch ( number ) {
case 10 : cout << “Running case 10 \n” ;  break;
case 20 : cout << “Running case 20 \n” ;  break;
case 30 : cout << “Running case 30 \n” ;  break;
default :  cout << “Running default code \n” ;  break;
}
return 0;
}
```

Output:

Running case 20

Explanation:

The number value is 20; it will match case 20, so it will run the code in case 20.

For Loop

```
for( init, test-expression, increment) { // some code; }
```

“for loop” runs a block of code by specified number of times.

Example 2.4

```
#include <iostream>
using namespace std;
int main( ) {
int x;
for (x = 0; x <= 5; x++){
cout << x;
}
return 0;
}      (Output: 012345 )
```

Explanation:

x = 0 is initializer,

x <= 5 is a test-expression, the code will run at most 5 times.

x++ means that x will increase 1each loop.

After 5 times loop, the code will output 012345.

While Loop

```
while ( test-expression ) {  // some C++ code in here;
}
```

“while loop” loops through a block of code if the specified condition is true.

Example 2.5

```
#include <iostream>
using namespace std;
int main( ) {
int counter=0;
while (counter < 8){
cout << “&”;
counter++;
}
return 0;
}      ( Output:  &&&&&&&& )
```

Explanation:

“counter< 8” is a test expression, if the condition is true, the code will loop less than 8 times, until the counter is 8, then the condition is false, the code will stop running.

Do-While Loop

```
do{ // some c++ code in here } while ( test-expression);
```

“do…while” loops through a block of code once, and then repeats the loop if the specified condition is true.

Example 2.6

```
#include <iostream>
using namespace std;
int main( ) {
int counter=0;
do {
cout <<  “@”;
counter++;
} while (counter<8);
return 0;
}    ( Output: @@@@@@@@ )
```

Explanation:

“counter< 8” is a test expression, if the condition is true, the code will loop less than 8 times, until the counter is 8, then the condition is false, the code will stop running.

Break Statement

```
Break;
```

“break” keyword is used to stop the running of a loop according to the condition.

Example 2.7

```
#include <iostream>
using namespace std;
int main( ) {
int num=0;
while ( num<10 ){
if ( num==5 )  break;
num++;
}
cout <<  num ;
return 0;
}    ( Output:  5)
```

Explanation:

“if (num==5) break;” is a break statement. If num is 5, the program will run the “break” command, the break statement will exit the loop, then run “cout << num ”.

Continue Statement

```
continue;
```

“continue” keyword is used to stop the current iteration, ignoring the following codes, and then continue the next loop.

Example 2.8

```
#include <iostream>
using namespace std;
int main( ) {
int num=0;
while ( num<10 ){
num++;
if ( num==5 )  continue;
cout <<  num ;
}
return 0;
} ( Output: 1234678910)
```

Explanation:

Note that the output has no 5.

“if (num==5) continue;” includes “continue” command. When the num is 5, the program will run “continue”, skipping the next command “cout << num ”, and then continue the next loop.

Boolean-Expression

If (bool-expression) { statements }
 	
while (boolean-expression) { statement }

The boolean-expression can be only a variable. For example: if (**var**) { }, if (**!flag**) { }.

Example 2.9

```
#include <iostream>
using namespace std;
int main( ) {
int num=10;
if ( num ) {    // equals “if (num == 10)”
cout << “num = 10” << endl;
}
while ( !num ){     // equals “while(num!=10)”
cout <<  “num !=10” <<endl;
}
return 0;
}    (Output:  num = 10 )
```

Explanation:

“if(**num**)” and “while (**!num**)” are correct codes, because they can have a variable inside the ( ).

In Java, it should be “if (num==10)” or “while (num**!**=10)”.

Conditional Operator

(test-expression) **?** (if-true-do-this) **:** (if-false-do-this);

(test-expression) looks like this: a<b, x!= y, m==n. etc.

Example 2.10

```
#include <iostream>
using namespace std;
int main( ) {
int a=100; int b=200;
string result = (a<b) ? “apple” : “boy”;
cout <<  result << endl;
return 0;
}      (Output:  apple)
```

Explanation:

The conditional operator use (a<b) to test the “a” and “b”, because “a” is less than “b”, it is true. Therefore, the output is “apple”.

“return 0;” indicates the program run successfully.

Exercises

For Loop Program

Write C++ codes to your favorite editor.

```
#include <iostream>
using namespace std;
int main(){
        int sum = 0;
    for (int n=1; n<=100; n++)
            sum += n;
    cout << “Sum = ” << sum << endl;
    return 0;
}
```

Save, Compile and Run the Program.

Output:

Sum = 5050

Hour 3

Use Array

Create an Array (1)

An array is a variable, which can contain multiple values.

The first method to create an array as following:

```
int array-name[ number] = {“value0”, “value1”, “value2”,…};
```

“number” means the number of array elements.

Example 3.1

```
int myarray[4] = { 10,20,30,40 };
```

Explanation:

The above code creates an array named “myarray”, which has four elements:

“myarray[4]” means that “myarray” has 4 elements

The 1st element is myarray[0] with value 10. Key is 0.

The 2nd element is myarray[1] with value 20. Key is 1.

The 3th element is myarray[2] with value 30. Key is 2.

The 4th element is myarray[3] with value 40. Key is 3.

In array, Key’s alias is “index”. Index’s alias is “key”.

Note that index begins from 0.

Create an Array (2)

An array is a variable, which can contain multiple values.

The second method to create an array as following:

```
int myarray [ number of elements ];
int array-name[index0] = “value1”;
int array-name[index1] = “value1”;
int array-name[index2] = “value2”;
```

Example 3.2

```
#include <iostream>
using namespace std;
int main( ) {
string color[3];
color [0] = “Red  “;
color [1] = “Yellow  “;
color [2] = “Green  “;
cout<< color[0]<<color[1]<<color[2];
return 0;
}
```

Output:

Red Yellow Green

Explanation:

Above code creates an array, array name is “color”, and it has three elements: color [0], color [1], color [2]. Its indexes are 0, 1, and 2. Its values are Red, Yellow and Green.

Note that index begins from 0.

Array Size

```
sizeof (array-name);
```

“sizeof (array-name);” can return the total number of bytes of an array.

Example 3.3

```
#include <iostream>
using namespace std;
int main( ) {
int arr[50];
int num = sizeof ( arr );
cout << num << ” bytes”;
return 0;
}     ( Output:  200 bytes )
```

Explanation:

“sizeof (arr)**;**” is used to get the length of the array “arr”, which means the total number of bytes.

Note that sizeof (datatype) can return the number of bytes of that data type. e.g. sizeof (int) returns 4 bytes.

Element Value

```
int value = array[index];


//  Get a value from an element.

int array[index] = value;


//  Set a value to an element.
```

Example 3.4

```
int arr[5] = {10, 20, 30, 40, 50};
int value = arr[2];
cout <<  value <<endl;
```

(Output: 30)

Example 3.5

```
int num[ 5 ];
num[3] = 800;
cout << num[3] <<endl;
```

(Output: 800)

Explanation:

“int value = arr[2];” gets a value 30 from “arr[2]”, and assigns to the “value” variable.

“num[3] = 800;” sets a value 800 to the element “num[3]” .

Function

Function is code block that can be repeated to use.

Before using a function, you need to declare a function first.

```
type function-name(  );   // declare a function
function-name(  );   // calls a function.
type function-name(  ) {…..};   // define a function
```

Example 3.6

```
#include <iostream>
using namespace std;
void myFunction( );   // declare a function
int main( ) {
myFunction( );    // call a function
return 0;
}

void myFunction ( ) {    // define a function
cout << “This is a function demo.” <<endl;
}
```

Output:

This is a function demo.

Explanation:

“void myFunction();” declares a function.

“myFunction( );” calls a function.

“void myFunction( ) {……}” defines a function.

“return 0” indicates the program run successfully.

“void” is a return type, meaning without return value. Namely without “return” command in function body.

Function & Arguments

A function sometimes has argument.

Before using a function, you need to declare a function.

```
type function-name( arg );   // declare a function
function-name ( argument );      // call function
type function-name ( arg ) {……};  // define function
```

Example 3.7

```
#include <iostream>
using namespace std;
void myFunction(string arg );   // declare a function

int main( ) {
myFunction ( “with arguments.” );   // call a function
return 0;
}

void myFunction ( string arg ) {  // define a function
cout << “This is a function ” + arg << endl;
}
```

Output:

This is a function with arguments

Explanation:

“void myFunction(string arg);” declares a function.

“myFunction(“with arguments”);” calls a function.

“void myFunction (string arg) {…}” defines a function with argument.

“myFunction(“with arguments”);” calls function “void myFunction(String argument) {…}”, and pass the argument “with arguments” to it. After receiving the argument “with arguments”, it outputs “This is function with arguments”.

Return Value

“return value” can return a value to the caller.

```
type function-name ( argument ) { return value };

function-name ( arg );
```

Example 3.8

```
#include <iostream>
using namespace std;
int myFunction(int arg );   // declare a function

int main( ) {
cout << “The number is ” << myFunction(10)<<endl;
return 0;
}

int myFunction ( int arg ) {
return ++arg;   // return a value to caller
}
```

Output:

The number is 11.

Explanation:

“myFunction(“10”)” calls a function. Therefore “myFunction(“10”)” is a caller.

“int myFunction (int arg) {…}” defines a function.

“int” is a return type, it means the function returns int type.

“return ++arg; ” can return the value “11” to the caller.

The output is “The number is 11”.

Call Another Function

```
function1 ( );
type function1 ( ) {  function2(  );  };
type function2 ( ) {  };
```

In function1, the function2( ) calls another function.

Example 3.9

```
#include <iostream>
using namespace std;
void firstFunction(int num);     // declare a function
int secondFunction(int arg );   // declare a function

int main( ) {
int num = 2;
firstFunction ( num );
return 0;
}

void firstFunction ( int num ){
cout << “The result is “<< secondFunction (num) <<endl;
}

int secondFunction( int num) {
return ( num * 50);
}
```

Output:

The result is 100.

Explanation:

In “firstFunction”, the “secondFunction (num)” calls a function named “int secondFunction(int num) { }”, and gets a return value 100.

Constants

Data that cannot change while running of the program is called *constants*.

```
#define CONSTANT-NAME  “value”
```

“#define” defines a constant.

“CONSTANT-NAME” is a constant name, using capital.

NOTE: Using #define, the sentence cannot have semicolon at the end.

Example 3.10

```
#include <iostream>
using namespace std;
#define SYMBOL  “$”
#define PRICE  80
#define  MONEY ” dollars.”
int main( ) {
int num = 10;
cout << SYMBOL << num * PRICE << MONEY;
return 0;
}
```

Output:

$800 dollars.

Explanation:

#define SYMBOL “$” defines a constant named “SYMBOL”, its value is “$”.

The PRICE value is “80”, the MONEY value “dollars”.

Data Type Conversion

```
variable 1= ( data-type) variable 2;
```

(data-type) can convert a data type to another data type.

Example 3.11

```
#include <iostream>
#define PI 3.1415926  // macro definition
using namespace std;
int main( ){
int number1, number2;
float  decimalNumber = 28.699;
bool  boolValue = false;
number1 = (int) decimalNumber;
number2 = (int) boolValue;
cout << “number1:  ” << number1 <<endl;
cout<<  “number2:  ” << number2 <<endl;
return 0;
}
```

Output:

Number 1: 28

Number 2: 0

Explanation:

(int) can change the data type to integer.

(float) can change the data type to float pointing number.

(string) can change the data type to string.

And so on…..

Exercises

Array Operation Program

Write C++ codes to your favorite editor.

```
#include <iostream>
using namespace std;
int main(){
    int arr[5]= {20, -6, 0, 100, 78};
    int max = arr[0];
    for(int n=1; n<5; n++)
                if(arr[n]>max)
                            max=arr[n];
    cout << “Max Value = “<< max << endl;
    return 0;
}
```

Save, Compile and Run the Program.

Output:

Max Value = 100

Hour 4

String

A String Variable

Before using string data type, <string> class should be included to the document first, which help using string.

```
#include <string>
```

To declare and initialized a string variable:

```
string  mystring = “some characters”
```

Example 4.1

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
string mystring = “We Love C++”;
cout << mystring << endl;
return 0;
}       ( Output:  We Love C++ )
```

Explanation:

“#include <string>” includes <string> class from C++ library.

“string mystring = “We Love C++” declares and initialize a string variable.

Input String Data

```
cin  >> string variable;
```

“cin >> string variable;” can input a single word.

Example 4.2

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
string mystring;    // declare a string
cout << “Please enter a word: “<< endl;
cin >> mystring;   // assume entering GOOD
cout <<  mystring  <<endl;
return 0;
}      ( Output:  Please enter a word:    GOOD )
```

Explanation:

“#include <string>” include “string” class.

“cin >> mystring;” can accept a word input from user.

Note that “cin >> mystring;” cannot input a whole line of sentence.

Input String Sentence

```
getline ( cin, mystring);
```

“getline (cin, mystring);” can let user input a string sentence.

Example 4.3

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
string mystring;
cout << “Please enter a sentence: ” << endl;
getline ( cin, mystring);    //  enter “C++ is Very Good!”
cout <<  mystring << endl;
return 0;
}       ( Output: Please enter a sentence:
    C++ is Very Good!)
```

Explanation:

“getline (cin, mystring);” accepts the user input by sentence. When a user enters “C++ is Very Good!”, the value of “mystring” is “C++ is Very Good!”.

Test Whether Inputted

```
mystring.empty( );
```

“mystring.empty( );” can test whether a user has inputted a word or sentence before submitting data. If mystring is empty, it returns true. Otherwise, it returns false.

Example 4.4

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
string mystring;
if ( mystring.empty( )) {
cout << “Please enter your name: “;
getline ( cin , mystring);   // assume entering My Name
cout << mystring << endl;
}
else {
cout << “You have entered your name, thank you! “;
}
return 0;
}
```

Output:

Please enter your name:

My Name

Explanation:

“if (mystring.empty( ))” can check the user whether has entered data or not. If true, it indicates the user has not entered any data.

String Length

```
mystring.size(  );
```

“mystring.size( )” can check the length of a string, returns the total number of characters and space in this string.

Example 4.5

```
#include<string>
#include <iostream>
using namespace std;
int main( ) {
string mystring = “operating system”;
int stringSize = mystring.size( );
cout << “String length is: ” << stringSize << endl;
return 0;
};      ( Output:  String length is: 16 )
```

Explanation:

“mystring.size( )” returns the size of “operating system”, the result is 16.

“return 0” indicates the program run correctly.

Note that a single space equals the string size of 1 character.

Find a Character

```
mystring. at(index);
```

“mystring.at (index)” gets a character from a string by index. The index of a string begins with 0.

Example 4.6

```
#include<string>
#include <iostream>
using namespace std;
int main( ) {
string mystring = “operating system”;
char aCharacter = mystring.at(2);
cout << “The character is: ” << aCharacter << endl;
return 0;
};     ( Output:  The character is:  e )
```

Explanation:

“mystring.at(2)” gets a character at index 2 from the string “operating system”.

Note that the index of a string begins with 0.

Connect Strings

```
originalString.append ( newString );


originalString + newString;
```

“append( )”can connect a new string to an existing string.

“+” can connect two strings together.

Example 4.7

```
#include<string>
#include <iostream>
using namespace std;
int main( ) {
string a= “Java”, b= “Script”, newWord, allWord;
newWord = a + b;
cout << newWord <<endl;   //Output: JavaScript
allWord = a.append(b);
cout << allWord << endl;    //Output: JavaScript
return 0;
};     (Output:   JavaScript      JavaScript)
```

Explanation:

“a + b” and “a.ppend (b)” concatenate “Java” and “Script” to a new String “JavaScript”.

Exchange Strings

```
string1.swap(string2);
```

“swap( )” can exchange two strings’ values.

Example 4.8

```
#include<string>
#include <iostream>
using namespace std;
int main( ) {
string s1 = “Number One.”;
string s2 = “Number Two.”;
cout << “string1: ” << s1<< endl;
cout << “string2: ” << s2 << endl;
s1.swap(s2);
cout << “string1: ” << s1<< endl;
cout << “string2: ” << s2 << endl;
return 0;
};
```

Output:

```
string1: Number One.
string2: Number Two.
string1: Number Two.
string2: Number One.
```

Explanation:

“s1.swap(s2);” swaps two strings’ value.

Find a Word’s Position

```
mystring.find ( substring, start )
```

“find( )” can locate a substring’s place.

“substring” means a substring that need to search.

“start” means a start index to find a substring.

Example 4.9

```
#include<string>
#include <iostream>
using namespace std;
int main( ) {
string mystring = “An operating system.”;
string substring = “system”;
int position = mystring.find ( substring, 0 );
cout << “The position of substring is at: ” << position << endl;
return 0;
}; (Output: The position of substring is at: 13 )
```

Explanation:

“system” locates index 13 in string “An operating System”. Note that string index begin with 0. One space in string equals one character size.

Insert a Substring

```
originalString = insert ( index, substring );
```

“originalString = insert (index, substring);” can insert a substring into an original string.

“index” means a position where inserting a substring.

Example 4.10

```
#include<string>
#include <iostream>
using namespace std;
int main( ) {
string originalString = “Flower is beautiful!”;
string substring  = “very “;
originalString.insert ( 10, substring);
cout << originalString << endl;
return 0;
};  ( Output:  Flower is very beautiful! )
```

Explanation:

“originalString.insert (10, substring);” inserts a substring “very” into an original string “Flower is beautiful” at the position 10.

Exercises

String Operation Program

Write C++ codes to your favorite editor.

```
#include <string>
#include <iostream>
using namespace std;
int main(){
string myString;
cout << “Please enter a sentence,” << endl;
cout << “then Enter!” << endl;
getline(cin, myString);
cout << “You have inputed: ” << myString << endl;
return 0;
}
```

Save, Compile and Run the Program.

Input “**Java in 8 Hours**”, and “Enter”, you can see the output:

Output:

You have inputted**:** Java in 8 Hours

Hour 5

Class & Object

Class Definition

A class is a template for object, and can create an object. Class represents one kind of something. Object represents a concrete thing.

```
class ClassName {    // define a class
public:              // specify the access permission
type variable;     //  define a variable
type function-name( ) { };    // define a function
};
```

Example 5.1

```
class Color {       // create a class named Color
public:    // specify a access permission
string c1, c2;
void brightness ( ) {  cout << “blue” <<endl; };
};
```

Explanation:

Above codes defines a class named “**Color**”, access permission “public”, two variables named “c1”, “c2” and a function named “brightness”.

Variables and method in a class are called as “**members**”.

Object Declaration

Object is an instance of a class.

```
ClassName  ObjectName;

ObjectName.variable;

ObjectName.function-name( );
```

“ClassName ObjectName” creates a new object for the class.

“ObjectName**.**variable;” means that “ObjectName” references a variable.

“ObjectName**.** function-name ( );” means that “ObjectName” calls a function.

Example 5.2

Color **Tint**; // create an object named “Tint”.

Tint.c1= “yellow”; Tint.c2=“purple”;

Tint**.**brightness ( ); // brightness( ){ } in previous page.

Explanation:

“Color **Tint**;” creates an object named “**Tint**”, then Tint references variable “c1” and “c2”.

“Tint.brightness ( );” means that an object “Tint” calls a function “brightness ( );”

Class & Object

```
class ClassName{  };    // define a class
ClassName  ObjectName;   // create an object
```

Example 5.3

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

class Color {       // create a class named Color
public:    // specify a access permission
string c1, c2;
void brightness ( ) {  cout << “Green” <<endl; };
};

int main( ) {
Color Tint;     // create an object named “Tint”.
Tint.c1= “Red “;   Tint.c2=“Yellow “;
cout << Tint.c1 << endl;
cout << Tint.c2 << endl;
Tint.brightness ( );
return 0;
}
```

Output:

Red

Yellow

Green

Explanation:

“class **Color**” defines a class “Color”.

“Color **Tint;**” creates an object “Tint”.

“Tint.c1” means an object “Tint” references a variable “c1”.

“Tint.brightness ( );” means an object “Tint” calls a method “brightness( ).”

Constructor

Constructor is used to initialize variables. Constructor name is the same as class name.

```
class ClassName{

ClassName( ) { …} ;     //  this is a constructor

};
```

class ClassName{ …} defines a class.

ClassName( ) { …} declares a constructor.

Example 5.4

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

class Color {       // create a class named Color
public:    // specify a access permission
string c1, c2;
void brightness ( ) {  cout << “Green” <<endl; };
Color ( ) {  c1=“Red”;  c2=“Yellow”; };  // constructor
};

int main( ) {
Color Tint;     // create an object named “Tint”.
cout << Tint.c1 << endl;
cout << Tint.c2 << endl;
Tint.brightness ( );
return 0;
}
```

Output:

Red

Yellow

Green

Explanation:

“**Color ( )** { c1=”yellow”; c2=”purple”; }” is a constructor. It initializes variable c1 as “yellow” and c2 as “purple”.

Destructors

Destructor is used to release the memory occupation, clean up the constructor data.

```
~ destructor( );
```

Destructor name is the same as class name, but it is preceded by a ~ symbol.

Destructor has no any arguments, and has no return value.

Example 5.5

```
class Color {  // define a class
public:
string c1; var c2;
Color ( ) {  c1=”yellow”;  c2=”purple”; } ;  // constructor
~ Color ( );     //  This is a destructor.
};
```

Explanation:

“~ Color ( );” is a destructor.

The declaration of destructor must follow the constructor.

Inheritance

A class can be inherited by its derived class. The object in derived class can be treated as the object in base class.

derived class inherits base class as following:

```
derived class: public base class
```

Example 5.6

```
class Computer   // this is a base class
{
// base class member…
};
class Laptop: public Computer    // inheritance
{
// derived class member…
};
```

Explanation:

“Laptop” is a derived class, “Computer” is a base class.

“Laptop” class inherits “Computer” class.

Derived class inherits public member in base class.

Using Derived class

An object of derived class can access the public member of base class.

Example 5.7

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

class Computer {   //  declare a base class
public:
void display( ){
cout << “Computer OK!” << endl;
}
} ;
class Laptop: public Computer {   // inheritance
// derived class’s member……};
};

int main ( ){
Laptop Lt;   // derived class creates an object Lt
Lt. display( );  // Lt calls base class’s method
return 0;
}
```

Output:

Computer OK!

Explanation:

“Lt**.** display( );” means a derived class object “Lt” calls base class’s method display( ).

Public Permission

“public” is one of the “Member Access Specifier”,.

public class members can be accessed by the object in current class or other class

Example 5.8

```
#include<string>
#include <iostream>
using namespace std;
class Computer {
public:    // define public permission
void display( ){
cout << “Computer OK!” << endl;
}
} ;
int main( ){
Computer obj;   //  create an object
obj.display( );
return 0;
}
```

Output:

Computer OK!

Explanation:

“obj.display( );” indicates an object “obj” calls “display( ){ }”.

Note that “obj” is outside the class “Computer”, but “obj” can access the public member “display( ){ }”.

Private Permission

“private” is one of the “Member Access Specifier”,.

private class member can be accessed by the object in current class but not in derived class, not in other class

Example 5.9

```
#include<string>
#include <iostream>
using namespace std;
class Computer {
private:    // define private permission
void display( ){
cout << “Computer OK!” << endl;
}} ;
int main( ){
Computer obj;   //  create an object
obj.display( );    // error
return 0;
}     (  Output:  Error Messages  )
```

Explanation:

“ obj.display( );” indicates an object “obj” calls “display( ){ }”.

Note that “obj” is outside the class “Computer”, so “obj” cannot access the private member “display( ){ }”.

Protected Permission

“protected” is one of the “Member Access Specifier”,.

protected class member can be accessed by the object in current class and derived class, but not in other class.

Example 5.10

```
#include <iostream>
using namespace std;
class A {
protected:
int prot;   // declare a protected member
};

class B:A {    // inheritance
public:
void myfunction(){
prot = 100; // access protected member
cout << prot;
}
};

int main(){
B obj;
obj.myfunction();
return 0;
}
```

Output:

100

Explanation:

“protected**:** int prot;” declares a protected member.

“prot = 100;” access protected member of the base class, because the protected member in base class can be accessed by derived class.

Class Method

Sometimes a method needs to be defined outside a class.

```
ClassName:: method( ) {  }
```

:: is used to specify the method belonging to a class.

Example 5.11

```
#include<string>
#include <iostream>
using namespace std;
class Time {
public:
void setHour( int);
int hour;
};
void Time::setHour(int h){    // class method
int hour = h;
cout << “The hour is : ” << hour << endl;
}

int main( ){
Time obj;   //  create an object
obj.setHour(10);
return 0;
}
```

Output:

```
The hour is:  10
```

Explanation:

“void Time**::**setHour(int h){ }” defines a method outside the class. It means that “Time” class has a method “setHour(int h){ }”.

“int hour = h;” means a variable “h” access a public member “hour”.

Access Private Member

Private Member is only accessed in current class.

Example 5.12

```
#include<string>
#include <iostream>
using namespace std;
class Time {
public:
void setHour(int);
private:
int hour;  // hour is a private member
};
void Time::setHour(int h){
int hour = h;   //“h” accesses private member “hour”, valid!
cout<< hour;
}


int main( ){
Time obj;   //  create an object
obj.setHour(10);
return 0;
}
```

Output:

10

Explanation:

“void Time**::**setHour(int h){ }” defines a method outside the class. It means “setHour( ){ }” belongs to the class “Time”.

“int hour = h;” means a variable “h” access a private member “hour”. It’s valid.

Exercises

Class & Object Program

Write C++ codes to your favorite editor.

```
#include<iostream>
using namespace std;
class SetTime{
public:
    int hour;
    int minute;
    int second;
};

int main(){
SetTime timeObject;
cout << “Please enter Hour, and Enter. ” << endl;

cin >> timeObject.hour;
cout << “Please enter Minute, and Enter.” << endl;

cin >> timeObject.minute;
cout << “Please enter Second, and Enter.” << endl;

cin >> timeObject.second;

cout << “You have set time as following:” << endl;
cout << timeObject.hour << “:”
    <<timeObject.minute <<”:”
    <<timeObject.second <<endl;
return 0;
}
```

Save, Compile and Run the Program.

Input “**Hour, Minute, Second**”, and “Enter”, you can see the output:

Output:

Please enter Hour, and Enter.

10

Please enter Minute, and Enter.

26

Please enter Second, and Enter.

38

You have set time as following**:**

10:26:38

Hour 6

Pointer & Reference

Pointer

Pointer is a variable that stores the memory address of the variable. Pointer name is preceded by a “ * ”.

```
dataType  *PointerName;
pointerName = &variable;
```

“dataType *PointerName ” defines a pointer.

“pointerName = &variable;” stores address of the variable into pointer. “&” means memory address.

Example 6.1

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
int num = 10;
int *numPointer ;        //  defines a pointer
numPointer = &num;   // stores address of num to pointer
cout << “Address of num is: ” << numPointer << endl;
cout << “Value of num is: ” << *numPointer <<endl;
return 0;
}
```

Output:

Address of num is: 0x22ff58

Value of num is: 10

Explanation:

“numPointer” represents the address of “num”.

“*numPointer” represents the variable of “num”.

Pointer Initialize

```
dataType *PointerName = &variable;
```

“dataType *PointerName = &variable” declares and initializes a pointer, stores the memory address of the variable into *PointerName”. “&” means memory address.

Example 6.2

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
int digit = 20;
int *digitPointer = &digit;
cout << “Address of digit is: ” << digitPointer << endl;
cout << “Value of digit is: ” << *digitPointer << endl;
return 0;
}
```

Output:

Address of digit is: 0x22ff58

Value of digit is: 20

Explanation:

“int *digitPointer = &digit;” declares and initializes a pointer. “digitPointer”, stores the memory address of “digit”

“digitPointer” represents the address of “digit”.

“*digitPointer” represents the variable of “digit”.

Using Pointer

“int *pointer1 = &x” means that *pointer1 is x variable.

“int *pointer2 = &y” means that *pointer2 is y variable.

Example 6.3

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
int x =10, y;
int *pt1 = &x, *pt2 = &y;
// declare & initialize two pointers
*pt2 = *pt1;   // namely  y = x;
*pt1= 20;      // namely  x = 20;
cout << x << ” ” << y <<endl;
cout << *pt1 << ” ” << *pt2 << endl;
return 0;
}
```

Output:

20 10

20 10

Explanation:

“*pointer” represents a variable when running program.

“int *pt1 = &x” means that *pt1 is “x” variable.

“int *pt2 = &y” means that *pt2 is “y” variable.

“*pt2 = *pt1;” namely “y=x”.

“*pt1= 20;” namely “x=20”.

Exchange Pointers

```
pt=pt1; pt1=pt2; pt2=pt;
```

“pt1” exchanges “pt2” in above codes.

Example 6.4

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

int main( ) {
int *pt, x, y;
int *pt1 = &x;     // *pt1 is x variable
int *pt2 = &y;     // *pt2 is y variable
x= 100, y = 200;
cout << x << ” ” << y << endl;
pt=pt1; pt1=pt2; pt2=pt;  // exchange pointers
cout << *pt1 << ” ” << *pt2 << endl;
return 0;
}
```

Output:

```
100   200
200   100
```

Explanation:

“*pt” represents a variable when running program.

Before exchanging pts, *pt1 represents x, *pt2 represents y.

After exchanging pts, *pt1 represents y, *pt2 represents x.

Pointer and Array

```
*pointer = array;
```

“*pointer = array;” means that the pointer points to **the first** element of the array

Example 6.5

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

int main( ) {
int myarray[ 3 ] = { 10, 20, 30 };
int *pt = myarray;
// pointer points to the first element myarray[0]
cout << “The first value is: ” << *pt << endl;
pt++;  // move pointer to next element myarray[1]
cout << “The second value is: ” << *pt << endl;
pt++;  // move pointer to next element myarray[2]
cout << “The third value is: ” << *pt << endl;
return 0;
}
```

Output:

The first value is: 10

The second value is: 20

The third value is: 30

Explanation:

“int *pt = myarray;” means that the pointer points to the first element of “myarray”. Namely myarrray[0].

“pt++” moves the pointer to next element.

“*pt” represents the array element.

Pointer Array

```
int a, b, c;
int *pointer[n] = { &a, &b, &c };
```

“int *pointer[n] = { &a, &b, &c };” declares a pointer array.

Example 6.6

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

int main( ) {
int a, b, c;
a = 10, b = 20, c = 30;
int *myarray[3] = { &a, &b, &c };
cout << *myarray[0] + *myarray[1] << endl;
cout << *myarray[2] - *myarray[1] << endl;
return 0;
}
```

Output:

30

10

Explanation:

“int *myarray[3] = { &a, &b, &c };” declares a pointer array.

*myarray[0] represents “a” variable.

*myarray[1] represents “b” variable.

*myarray[2] represents “c” variable.

Pointer & String

```
char *pointer = “mystring”;
```

“char *pointer = “mystring”;” defines a pointer which points to **the first** character of the “mystring”.

Example 6.7

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

int main( ) {
char *pt = “mystring”;
int counter = 0;    // define a counter
while ( *pt!=0 ) {  // pointer points to a character
counter++;   // counts how many characters
pt++;    // pointer moves to the next character
}
cout << counter;  // output the number of characters.
return 0;
}
```

Output:

8

Explanation:

“char *pt = “mystring”;” make the pointer points to the first character of “mystring”.

*pt**!**=0 means that the pointer does not points to nothing.

“pt++” moves the pointer points to the next character in “mystring”.

Reference a Variable

Reference is an **alias** for a variable or an object. A reference name is preceded by a symbol “&”.

```
dataType &reference = variable;
```

“&reference” is an alias for the variable.

Example 6.8

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

int main( ) {
int num;
int &r = num;   // “r” is an alias of “num”
r = 100;
cout << num << endl;
num = 200;
cout << r << endl;
return 0;
}
```

Output:

100

200

Explanation:

“int &r = num” defines that the alia of “num” is “r”.

“r = 100;” indicates “num = 100”.

“num = 200;” indicates “r = 200”.

Reference an Object

```
ClassName &reference = object
```

“&reference” is an alias for the object.

Example 6.9

```
#include <string>
#include <iostream>
using namespace std;
class Color{
public:
void display();
};
void Color:: display( ){
cout <<“The color is blue.” << endl;
}

int main( ) {
Color  Tint;  // create an object”Tint”
Color &r = Tint;   //“r” is an alias of”Tint”
r.display( );
return 0;
}
```

Output:

The color is blue.

Explanation:

“Color &r = Tint;” references an object “Tint”, whose alias is “r”.

“r.display( );” means that a referenced object can calls a method “display( ){ }”.

Reference Arguments

```
dataType functionName ( &a, &b ) {  }
```

“functionName (&a, &b)” means that function arguments can be referenced.

Example 6.10

```
#include <string>
#include <iostream>
using namespace std;
void swap ( int &a, int &b) {
int temp;
temp = a, a = b, b = temp;   //  reference exchange
}

int main( ) {
int x = 10, y = 20;
cout << x << ” ” << y << endl;
swap ( x, y );
cout << x << ” ” << y << endl;
return 0;
}
```

Output:

10 20

20 10

Explanation:

“void swap (int &a, int &b)” uses reference in argument.

Before swap, “a” is an alias of “x”, “b” is an alias of “y”.

After swap, “a” is an alias of “y”, “b” is an alias of “x”.

Exercises

Pointer Sample Program

Write C++ codes to your favorite editor.

```
#include<iostream>
using namespace std;
int main( ){
int x=100, y=200;
int *xPointer = &x;
int *yPointer;
yPointer=&y;
cout<<“X value: “<<*xPointer<<endl;
cout<<“X address: “<<xPointer<<endl;
cout<<“Y value: “<<*yPointer<<endl;
cout<<“Y address: “<<yPointer<<endl;
return 0;
}
```

Please save, compile and run the program.

Output:

X value: 100

X address: 0x22ff54

Y value: 200

Y address: 0x22ff50

Hour 7

File Operation

Output One Character

```
cout.put (char C)
```

“cout.put (char C)” outputs one character.

Example 7.1

```
#include <string>
#include <iostream>
using namespace std;
int main( ) {
char ch = ‘C’;     // Note: use single quote.
cout.put (ch) << endl;
return 0;
}
( Output:  C  )
```

Explanation:

“#include <iostream>” includes iostream class that helps to input or output.

“cout**.**put (ch)” display one character “C”.

Output String

```
cout.write (stringArray, stringLength);
```

“cout.write( )” can output a string.

“stringArray” defines the string as an array.

“stringLength” defines the string size.

Example 7.2

```
#include <iostream>
#include <string.h>
using namespace std;
int main(){
char ch[ ]=“This is a test”;
cout.write(ch, strlen(ch))<<endl;
return 0;
}      ( Output:  This is a test )
```

Explanation:

“ch[ ] = “This is a test” defines a string array.

“strlen (ch)” gets the size of ch.

cout**.**write ( ) is used to output a string.

Input One Character

```
cin.get (char c );
```

“cin.get (char c);” is used to input one character by keyboard.

Example 7.3

```
#include <iostream>
#include <string.h>
using namespace std;
int main(){
char c;
cout << “Input one character: “<< endl;
cin.get ( c );
cout << c <<endl;
return 0;
}
```

(Output Input one character:)

Explanation:

“cin**.**get (c);” inputs one character “A” by keyboard.

“cout << c <<endl;” displays what have inputted.

Input String

```
cin.read ( stringArray, stringLength);
```

“cin**.**read ( )” is used to input strings.

“stringArray” defines the string as an array.

“stringLength” defines the string size.

Example 7.4

```
#include <iostream>
#include <string.h>
using namespace std;
int main(){
cout<< “Please input a sentence:”<< endl;
char mystring[ 80];   // max length is 80
cin.read ( mystring , 10 );
cout << “What you inputed is: ” << mystring << endl;
return 0;
}       ( Output:  Please input a sentence: )
```

Explanation:

“char mystring[80];” defines a string array, whose maximum size is 80 characters.

“cin**.**read (mystring, 10)” inputs a sentence with 10 characters.

Input String Sentence

```
getline ( cin, mystring);
```

“getline (cin, mystring);” can let user input a string sentence.

Example 7.5

```
#include <iostream>
#include <string.h>
using namespace std;
int main(){
string mystring;
cout << “Please enter a sentence:” << endl;
getline ( cin, mystring );    //  enter “C++ is very good”
cout <<  mystring << endl;
return 0;
}       ( Output: Please enter a sentence:
                    C++ is very good    )
```

Explanation:

“getline (cin, mystring);” accepts the user input by sentence. When a user enters “C++ is very good”, the value of “mystring” is “C++ is very good”.

Write a File

For operating file, you need to use #include <fstream>.

```
ofstream fileObject (“myFile.txt”);
fileObject << mystring << endl;
```

“ofstream fileObject (“myFile.txt”);” creates an output file object “fileObject”.

“ofstream” class is used to write data to a file.

“fileObject” is an output file object.

“fileObject << mystring << endl;” writes mystring to file.

Example 7.6

// Please create an empty file “myFile.txt” in the Project Folder first.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(){
ofstream fileObject ( “myFile.txt” );
string mystring = “C++ is a very good language.”;
fileObject << mystring << endl;
fileObject.close ( );
return 0;
}
```

Explanation:

“ofstream” class is used to write data to a file.

“ofstream fileObject (“myFile.txt”);” creates an output file object “fileObject”.

“fileObject << mystring << endl;” writes mystring to the file “myFile.txt”.

In myFile.txt, you will find some texts as following:

C++ is a very good language.

Open a File

When opening a file, you can specify the “File Mode”.

Mode	Usage
ios :: in	open a file to read input
ios :: out	open a file to write output
ios :: app	open a file to append output
ios :: trunc	open a file or truncate old file
ios :: ate	open a file without truncating old file
ios :: binary	open a file as binary file

Example 7.7

```
ofstream fileObject ( “myfile.txt”, ios :: app );  //line1
ofstream fileObject ( “myfile.txt”, ios :: trunk  );  //line2
ofstream fileObject ( “myfile.txt”, ios :: binary );  //line3
```

Explanation:

Line1: Open “myfile.txt”, append the output at the end of existing content.

Line2: Open “myfile.txt”, or truncate the existing file, and clear up all data.

Line3: Open “myfile.txt”, treat the file as binary format instead of text format.

Read a File

For operating file, you need to use #include <fstream>.

```
ifstream fileObject ( “myFile.txt” );
getline ( fileObject, mystring);
```

“ifstream fileObject (“myFile.txt”);” creates an input file object “fileObject”.

“ifstream” class is used to read data from a file.

“fileObject” is an input file object.

“getline (fileObject, mystring);” reads file to mystring.

Example 7.8

```
// Please create a file “myFile.txt” in the Project Folder first.
// “myFile.txt” has content: “C++ is a very good language.”

#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(){
string mystring;
ifstream fileObject ( “myFile.txt” );
getline ( fileObject, mystring );
cout << mystring <<endl;
fileObject.close ( );
return 0;
}
```

Output:

C++ is a very good language.

Explanation:

“ifstream” class is used to read data from a file.

“ifstream fileObject (“myFile.txt”);” creates an input file object “fileObject”.

“getline (fileObject, mystring);” reads the file to mystring.

End of File

```
fileObject.eof ( );
```

“fileObject.eof( );” means that while reading data, eof( ) returns true at the end of file, otherwise eof( ) returns false at somewhere of file.

Example 7.9

```
// “myFile.txt” has content: “C++ is a very good language.”

#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(){
char ch;
ifstream fileObject ( “myFile.txt”);
while ( ! fileObject.eof( ) ) { // not at the end of file
fileObject.get (ch);
cout << ch;
}
fileObject.close ( );
return 0;
}
```

Output:

C++ is a very good language.

Explanation:

“**!**fileObject**.**eof( )” indicates that “not at the end of file while reading data.”

“fileObject**.**get (ch);” read characters from “myfile.txt”

“fileObject**.**close( )” closes the file.

Exercises

Read a File

(1) Please prepare **myFile.txt** as following:

(myFile.txt)

```
PHP in 8 Hours!
JAVA in 8 Hours!
JQUERY in 8 Hours!
JAVASCRIPT in 8 Hours!
```

Save **myFile.txt** in the same project folder with the following C++ file.

(2) Write C++ codes to your favorite editor.

```
#include<fstream>
#include<iostream>
using namespace std;
int main(){
char word;
ifstream fileObject("myFile.txt");
if(!fileObject){
    cout<<"File is not existing" << endl;
    return -1;
    }
else{
while(!fileObject.eof()){
    fileObject.get(word);
    cout<<word;
    }
fileObject.close();
return 0;
    }
}
```

Please save, compile and run the program.

Output:

PHP in 8 Hours!

JAVA in 8 Hours!

JQUERY in 8 Hours!

JAVASCRIPT in 8 Hours!

Hour 8

C++ Study

this -> member

this -> member

“this” represents a current object.

“this -> member” means that the “this” references a member variable or member method.

Example 8.1

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

class Time{
public:
int num;
void setHour(int num );
};
void Time::setHour( int num ){
this->num = num;   //“this” represents current object
cout << “The hour is: ” << num << endl;
}

int main() {
Time t;      // creates an object “t”
t.setHour(10);
return 0;
}
```

Output:

The hour is: 10

Explanation:

“this” represents object “t”.

Static Variable

Static variable can be referenced by a class or any objects. Non-static variable is referenced only by object, not by class.

```
static dataType variable;
className :: variable;
```

“static” declaration means that variable belongs to class. “className **::** variable;” means that a class can references this static variable directly.

Example 8.2

```
#include <iostream>
using namespace std;
class Account {
public:
static int num;   // static declaration
};
int Account :: num = 100;   // class references a variable
int main() {
cout << Account :: num << endl;
return 0;
}
```

Output:

100

Explanation:

"**static** int num" means a static variable "num" can be referenced by a class or an object.

"Account **::** num" means a class "Account" can references a static variable "num".

Non-static variable is referenced only by object, not by class.

Static Function

Static function can be referenced by a class or any objects. Non-static function is referenced only by object, not by class.

```
static dataType function( );
className :: function( );
```

“static” declaration means that function belongs to class. “className **::** function( );” means that a class can references this static function directly.

Example 8.3

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

class Account {
public:
static int myfunction( int n );   // static declaration
};
int Account :: myfunction(int n){
return n;
}

int main() {
cout << Account :: myfunction( 200 ) << endl;
return 0;
}
```

Output:

200

Explanation:

"**static** int myfunction(int n)" means a static function "myfunction(int n)" can be referenced by a class or an object.

"Account **::** myfunction (200)" means a class "Account" can references a static function "myfunction(200)".

Non-static function is referenced only by object, not by class.

A++ or ++A

++A	A plus 1 first, then run expression
A++	Run expression first, then A plus 1
- - B	B minus 1 first, then run expression
B - -	Run expression first, then B minus 1.

Example 8.4

```
int A = 10, num;
num = ++A - 2;
/* A plus 1equals 11, 11 minus 2 equals 9, then the result “9” assigns to num.*/
cout << num << endl;   //  (  output:  9 )
```

Example 8.5

```
int A = 10, num;
num = A++ - 2;
/* A minus 2 equals 8, the result “8” assigns to num, then A plus 1.*/
cout << num << endl;  // (  output:  8  )
```

Explanation:

“++A” increase 1 first. “A++” run the expression first.

“- -A” and “A- -” are the same usage as “++A” and “A++”.

Call My Own Function

A function body includes its own function with the same function name, which is known as a recursive function.

Example 8.6

```
#include <iostream>
using namespace std;
void myself(int n);

int main() {
myself ( 1 );
return 0;
}

void myself ( int n){
cout <<  “Number ” << n << endl;
++n;
if ( n == 4) return;
else myself ( n );     // call myself
}
```

Output:

Number 1

Number 2

Number 3

Explanation:

“void myself (int n){ }” body contains “myself(n);”

and “void myself (int n)” is called by “myself(n);”, which is known as recursive function.

“else myself (n);” calls “void mysef (int n)”.

Local & Global Variable

Local variable is defined inside the function.

Global variable is defined outside the function.

Local variable is only visible in current function.

Global variable is visible in any function.

Example 8.7

```
#include <iostream>
using namespace std;
void test1();
void test2();
int main(){
test1();
test2();
return 0;
}
int a = 100, b = 200;    //  global variable
void test1 ( ) {             // define a function
int x =30, y =40;          //   local variable
cout << a << “,” << b << endl;
cout << x << “,” << y << endl;
}
void test2 ( ) {              // define a function
int x =50, y =60;           //   local variable
cout << a << “,” << b << endl;
cout << x << “,” << y << endl;
}
```

Output:

100, 200

30, 40

100, 200

50, 60

Explanation:

“a” and “b” are global variables, which are visible in any function.

“x” and “y” are local variables, which are visible only in current function.

Exceptions

C++ program may have some bugs, which calls “Exceptions”.

The main exceptions as following:

Exceptions	**Situations**
domain_error	out of precondition
invalid_argument	invalid argument in function
length_error	size too long or short
out_of_range	invalid range
overflow_error	data overflow

Example 8.8

```
int main( ) {
int a, b=0;
a=100/b;    //  exception occurs
cout << a << endl;return 0;
return 0;
}
```

(Output: Exception: ArithmeticException: / by zero)

Explanation:

“100/b” causes an Exception, because “b” is zero.

Try-Catch

For “try-catch”, you need to use #include<stdexcept>.

```
try
{ …}
catch(exception)
{……}
```

“try {… …}” throws the exception out to the catch( ) { } block when an exceptional error occurs.

“catch( ){ }” catches any exception from try block, and handles it.

Example 8.9

```
#include <iostream>
#include <stdexcept>
using namespace std;
int main(){
string str1=“JavaScript”;
string str2=“and”;
try{
str1.insert(100, str2);  // “100” causes error
}
catch(logic_error){
cout << “Catch an exception.” << endl;
cout << “Out of range!” << endl;
}
return 0;
}
```

Output:

```
Catch an exception.
Out of range!
```

Explanation:

In try{ } block, “str1.insert(**100**, str2);” causes an exception.

“catch(logic_error){ }” catches the exception.

Return Exception Message

```
try
{…}
catch(exception& e)
{  cout  << e.what( )  <<endl;  }
```

“e.what( )” can return exception message.

Example 8.10

```
#include <stdexcept>
#include <iostream>
using namespace std;
int main ()
{
string str = “JavaScript”;
try{
str.at(100);
}
catch(exception& e) {
cout<<“Catch an exception: “<< e.what( ) << endl;
}
return 0;
}
```

Output:

Catch an exception**:** basic_string**::** at

Explanation:

“catch(exception& e)” can catch an exception. Note that its parameters is “exception &e”, “e” is an exception object.

“e.what( )” returns exception message.

Throw Exception

```
try
{… throw…}
catch
{……}
```

“throw” can throw out an exception to catch block that can handle the exception.

Example 8.11

```
#include <stdexcept>
#include <iostream>
using namespace std;
int main (){
try{
int a=100, b=0;
if(b==0){
throw b;
a=a/b;
}
}
catch (int e){
cout << “An exception occurred.” << endl;
cout << “The exception integer division by ” << e ;
}
return 0;
}
```

Output:

An exception occurred.

The exception integer division by 0

Explanation:

“throw b;” throws an exception from b. If b equals zero, a/b will be an error.

“catch (int e){ }” can catch and handle the exception.

Vector

Vectors are similar to arrays that can change in size. Just like arrays, vectors can be any data type and its first index starts with zero.

If use vector in coding, C++ <vector> library must be included by #include <vector> at the beginning of the program.

The syntax of creating a vector looks like this:

```
Vector <data-type> vector-name (size);
```

Example 8.12

```
vector <int> myVector (10);     // Line 1
vector <double> prices (60);   // Line2
vector <string> titles (18);       //  Line 3
```

Explanation:

Line 1: Declares a vector of 10 integer elements.

Line 2: Declares a vector of 60 double elements.

Line 3: Declares a vector of 18 string elements.

Vector.method( )

push_back(value)	add an element to the end of the vector
size( )	return the size of the vector
at(index)	return the value at specified index
pop_back( )	delete the last element

Example 8.13

```
#include <vector>
#include <iostream>
using namespace std;
int main (){
vector < int > myVector ( 1 );
myVector.push_back(20);
myVector.push_back(30);
cout<< “myVector size: ” << myVector.size( ) << endl;
cout<< “The third element: ” << myVector.at(2) << endl;
myVector.pop_back( );
cout<< “Final element has been removed.”<< endl;
return 0;
}
```

Output:

myVector size: 3

The third element: 30

Final element has been removed.

Explanation:

vector < int > myVector (1) creates a vector with 1 element.

myVector.push_back(20) adds an element with value 20 to the end of the vector.

myVector.push_back(30) adds an element with value 30 to the end of the vector.

myVector.size( ) gets the number of element in myVector.

myVector.at(2) gets the value whose index is 2.

myVector.pop_back( ) removes the final element in myVector.

Vector.method( )

front( )	return the value of first element
back( )	return the value of last element
clear( )	clear the vector
empty( )	return 1 if the vector is empty, or 0 otherwise

Example 8.14

```
#include <vector>
#include <iostream>
using namespace std;
int main (){
vector < int > myVector ( 1 );
myVector.push_back(20);
myVector.push_back(30);
cout<< “The first element: ” << myVector.front( ) <<endl;
cout<< “The last element: ” << myVector.back( ) <<endl;
myVector.clear( );
cout<< “myVector is empty? ” << myVector.empty( ) <<endl;
return 0;
}
```

Output:

The first element: 0

The last element: 30

myVector is empty? 1

Explanation:

vector < int > myVector (1) creates a vector with 1 element.

myVector.push_back(20) adds an element with value 20 to the end of the vector.

myVector.push_back(30) adds an element with value 30 to the end of the vector.

myVector.front( ) returns the value of the first element.

myVector.back( ) returns the value of the last element.

myVector.clear( ) clears myVector.

myVector.empty( ) tests myVector to see if is empty.

“1” represents true, “0” represents false.

Exercises

Exception Handling Program

Write C++ codes to your favorite editor.

```
#include<stdexcept>
#include<iostream>
using namespace std;
int main(){
        string myString1 = “Java in 8 Hours” ;
        string myString2 = “eBook”;
try{
        cout<<“Original: “<<myString1<<endl;
        myString1.insert(100,myString2);
        cout<<“New String: “<<myString1<<endl;
}
catch(out_of_range){
        cout<< “Exception: Out of range!”;
        return 0;
}
}
```

Please save, compile and run the program.

Output:

Original**:** Java in 8 Hours

Exception**:** Out of range!

Appendix 1
Advanced C++ (1)

Extern Variable

```
extern dataType variable;
```

“extern dataType variable” declares a variable that can be used in other file.

Example a1

(file1.cpp)

```
#include<iostream>
using namespace std;
extern int c;   //declare an extern variable
int main( ){
cout << c << endl;
return 0;
}
```

(file2.cpp)

```
#include <iostream>
using namespace std;
int a = 100, b = 200;
int c = a + b;
```

Output:

300

Explanation:

“**extern** int c;” declares a variable that can be used in other file.

Note: Two files should belong to the same project. One C++ project can consist of one or more files. (.cpp), but only one file has “main( ){ }” function.

Extern Function

```
extern functionName( );
```

“extern functionName( );” declares a function that can be called in other file.

Example a2

(file1.cpp)

```
#include<iostream>
using namespace std;
extern void min(int, int);  //declares an extern function
int c;
int main( ){
int a = 100, b = 200;
min(a, b);  // call a function in file2
cout << c << endl;
return 0;
}
```

(file2.cpp)

```
#include <iostream>
using namespace std;
extern int c;  // define an extern variable
void min(int x, int y){
c = ( x < y) ? x : y;
}
```

Output:

100

Explanation:

“extern void min(int, int)” declares a function that can be called in other file.

“extern int c;” declares an extern file than can be used in other file.

Note: Two files should belong to the same project. One C++ project can consist of one or more files. (.cpp), but only one file has “main( ){ }” function.

Static Global Variable

```
static dataType variable;
```

If “static dataType variable;” is defined outside a function, it will be a static global variable.

“static dataType variable;” declares a variable that can be used only in current file.

Example a3

(file1.cpp)

```
# #include<iostream>
using namespace std;
static int c;   //declare static global variable
int main( ){
int a = 30, b = 40;
c = a + b;
cout << c << endl;
return 0;
}
```

(file2.cpp)

```
#include <iostream>
using namespace std;
int a = 100, b = 200;
int c = a + b;
```

Output:

70

Explanation:

“static int c;” declares static global variable “c” only being used in current file, and cannot be used in other file.

Note:

If “static variable” is declared as a “static member”, please check Hour 8.

If “static variable” is declared as a “global variable”, please check this page example.

Static Local Variable

You may not want a variable destroyed when the function is exited, you can use “static” to declare a local variable.

```
static dataType variable;
```

If “static dataType variable;” is defined inside a function, it will be a static local variable.

Example a4

```
#include<iostream>
using namespace std;
void myFunction();
int main( ){
myFunction( );
myFunction( );
myFunction( );
return 0;
}
void myFunction(){
static int num = 1;   // define a static local variable
num = num + 2;
cout<< num << endl;
}
```

Output:

3

5

7

Explanation:

“**static** int num = 1” defines a static local variable, which can retain its output value to next call.

Note:

If “static variable” is declared as a “static member”, please check Hour 8.

If “static variable” is declared as a “global variable”, please check previous page example.

If “static variable” is declared as a “local variable”, please check this page example.

Static Function

```
static dataType functionName( )
```

“static dataType functionName( )” declares a function that can be called only in current file.

Example a5

(file1.cpp)

```
#include<iostream>
void min(int, int);
int c;
int main( ){
int a = 100, b = 200;
min(a, b);  // call a function
return 0;
}
```

(file2.cpp)

```
#include <iostream>
using namespace std;
extern int c;
static void min(int x, int y){  // declare a static function
c = ( x < y) ? x : y;
cout << c << endl;
}
```

Output:

Compile error!

Explanation:

“static void min(int x, int y)” declares a static function that can be called only in current file.

In file1.cpp, “min(a, b);” want to call a function in file2.cpp, but “**static** void min(int x, int y)” is a static function, which can be called only in current file. Therefore, the output is error.

Note:

If “static function” is declared as a “static member”, please check Hour 8.

If “static function” is declared as an “internal function”, please check this page example.

Overloading

Overloading means that there are two or more same-name methods in a class, and their arguments are different.

Example a6

```
#include <iostream>
#include <string>
using namespace std;
class  Flower{
public:
Flower(string name);
Flower(string name, int number);
Flower(string name, int number, string color);
};
int main( ){
Flower FlowerName(" Rose ");
Flower FlowerNumber(" Rose ", 10);
Flower FlowerColor(" Rose ", 10, " Red ");
return 0;
}
Flower::Flower(string name)
{cout << name << endl;}
Flower::Flower(string name, int number)
{cout << name << number << endl;}
Flower::Flower(string name, int number, string color)
{cout << name <<number << color << endl;}
```

Output:

Rose

Rose 10

Rose 10 Red

Explanation:

There are three constructor methods. Their name is the same “**flower**”, but the arguments are different.

“flower flowerName(“Rose”);” creates an object “flowerName” and call the first constructor “**flower**(string name)”.

“flower flowerNumber(“Rose”, 10);” creates an object “flowerNumber” and call the second constructor “**flower**(string name, int number)”.

“flower flowerColor(“Rose”, 10, “Red”);” creates an object “flowerColor” and call the third constructor “**flower**(string name, int number, string color)”.

Overriding

The method of derived class can override the method of base class, if two method names are the same and two arguments are the same.

Example a7

```
#include <iostream>
using namespace std;
class Computer{  // base class
public:
void harddrive( ){ cout << “5T” << endl; };
void memory( ){ cout << “4G” << endl; };
};
class Laptop: public Computer{     //derived class
public:
void harddrive( ){ cout << “10T” << endl; }
void memory( ){ cout << “8G” << endl; }
};
int main( ){
Laptop myLaptop; // creates an object
myLaptop.harddrive( );
myLaptop.memory( );
return 0;
}
```

Output:

```
10T
8G
```

Explanation:

The method name “harddrive( )” and “memory( )”in base class is the same as the one in the derived class. And the method’s argument in base class is the same as the one in derived class. Therefore, the method in derived class can override the method in base class.

“myLaptop**.**harddrive( );” calls the method in derived class, and override the method in base class. The output is “10T”.

“myLaptop**.**memory( );” calls the method in derived class, and override the method in base class. The output is “8G”.

Friend Function

If a function is defined as “friend function” of a class, then the “friend function” can access the private member of current class.

Example a8

```
#include<iostream>
using namespace std;
class Week{
public:
Week(int, int, int);
friend void show( Week &);  //define friend function
private:     // define private member
int Mon;  int Tue;  int Wed;
};
Week::Week(int m, int t, int w){   // constructor
Mon = m;  Tue = t;  Wed = w;
}
void show ( Week &wk ){  // this is a friend function
cout << wk.Mon<< endl;
cout << wk.Tue<< endl;
cout << wk.Wed<< endl;
}
int main ( ){
Week weekObject( 1, 2, 3 );  // create an object
show( weekObject );  // call friend function
return 0;
}
```

Output:

1

2

3

Explanation:

“**friend** void show(Week &);” declares a friend function. Argument “Week &” indicates the data type of argument is “Week”. “&” is a reference symbol.

“void show (Week &wk){…}” is a friend function. It can access the private variable “Mon”, “Tue”, and “Wed”.

“wk” is an alias of the “weekObject”.

Exercises

Static Local Variable

Write C++ codes to your favorite editor.

```
#include<iostream>
using namespace std;
void myFunction();
int main( ){
myFunction( );
myFunction( );
myFunction( );
}
void myFunction(){
static int num = 1;   // define a static local variable
num = num + 2;
cout<< num << endl;
}
```

Save, Compile and Run the Program

Output:

3

5

7

Exercises

Friend Function

Write C++ codes to your favorite editor.

```
#include<iostream>
using namespace std;
class Week{
public:
Week(int, int, int);
friend void show( Week &);  //define friend function
private:
int Mon;  int Tue;  int Wed;
};
Week::Week(int m, int t, int w){    // constructor
Mon = m;  Tue = t;  Wed = w;
}
void show ( Week &wk ){  // this is a friend function
cout << wk.Mon<< endl;
cout << wk.Tue<< endl;
cout << wk.Wed<< endl;
}
int main ( ){
Week weekObject( 1, 2, 3 );  // create an object
show( weekObject );  // call friend function
return 0;
}
```

Save, compile and run the program

Output:

1

2

3

Appendix 2

Advanced C++ (2)

Virtual Function

```
virtual dataType functionName( );
```

“virtual dataType functionName( );” define a virtual function. When the function is called, it executes the overriding function in the derived class, rather than the function in the base class.

Example b1

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

class Car{
public:
virtual void drive( ){  //define a virtual function
cout << “Drive a car.” << endl;
}};
class Van: public Car{
public:
void drive( ){
cout << “Drive a van.” << endl;
}};
class Bus: public Car{
public:
void drive( ){
cout << “Drive a bus.” << endl;
}
};
void myFunction(Car &c){   //Car’s alias is “c”.
c.drive( );  // call drive( ) in derived class
}
int main( ){
Van v;  // creates an object
Bus b;  // creates an object
myFunction(v);
myFunction(b);
return 0;
}
```

Output:

Drive a Van.

Drive a Bus.

Explanation:

“virtual void drive( )” defines a virtual function.

“c.drive( );” only calls the function in derived class, instead of the function in base class.

“Car &c” is an argument which receives the parameter “v” or “b”, then decides which function in derived class will be called.

Abstract Class

```
virtual dataType functionName( ) = 0;
```

“virtual dataType functionName( ) = 0;” defines a “pure virtual method”. The “pure virtual method” always is inside an “abstract class”.

Any class that contains a pure virtual function is named “abstract class”. Abstract class as a base class includes one or more “pure virtual function”, which must always be overridden in derived class.

Example b2

```
#include<iostream>
using namespace std;
class Ball{    //define an abstract class
public:
virtual void play( ) = 0;  //define a pure virtual function
};
class Volleyball: public Ball{
public:
void play( ){
cout << “Play volleyball.” << endl;
}
};
class Football: public Ball{
public:
void play( ){
cout << “Play football.” << endl;
}
};
void sport(Ball &b){    //Ball’s alias is b
b.play( );    //call play( ) in derived class
}
int main( ){
Volleyball v;   // creates an object
Football f;     // creates an object
sport(v);
sport(f);
return 0;
}
```

Output:

Play volleyball.

Play football.

Explanation:

“class Ball” defines an abstract class, because it contains a pure virtual function.

“virtual void play( ) = 0;” defines a pure virtual function, which will always be overridden.

“b.play( )” only calls the function in derived class, instead of the function in base class.

“Ball &b” is an argument which receives the parameter “v” or “f”, then decides which function in derived class will be called.

Inline Function

The inline functions can increase the execution efficiency of a program. Whenever the inline function is called, the compiler will replace the function call with the actual code from the function. The syntax looks like this:

```
inline dataType functionName( );
```

Example b3

```
#include <iostream>
using namespace std;
inline int sum(int n);  //define an inline function
int main( ){
int number;
cout << “Please enter a number: “;
cin >> number;
number = sum(number);
cout << “Result: ” << number << endl;
number = sum(number);
cout << “Result: ” << number << endl;
number = sum(number);
cout << “Result: ” << number << endl;
}
int sum(int n){
return n + n;
}
```

Output:

Please enter a number: 3

Result: 6

Result: 12

Result: 24

Explanation:

“inline int sum(int n);” defines an inline function.

Each assignment to the variable “number” has been compiled as “number = number + number”.

Function Template

One function template can substitute several functions' code.

If several function bodies are the same, only data types are different, you can use Function Template.

```
template <typename T>   dataType functionName( T a, T b)
```

"T" is a type parameter. It stands for different data type.

Example b4

```
#include <iostream>
using namespace std;
template <typename T> T min(T a, T b, T c){
if ( b < a ) a = b;
if ( c < a ) a = c;
return a;
}
int main( ){
int  n, n1 = 10, n2 = 20, n3 = 30;
double  d, d1 = 10.01, d2 = 10.02, d3 = 10.03;
long  g, g1= 67891, g2 = 67892, g3 = 67893;
n = min( n1, n2, n3 );
d = min( d1, d2, d3 );
g = min( g1, g2, g3 );
cout << n << endl;
cout << d << endl;
cout << g << endl;
return n + n;
}
```

Output:

```
10
10.01
67891
```

Explanation:

“T min(T a, T b, T c)” defines a function template “min( )”.

“T” can be various data type. It can be int, double, long.

“return a” can return different value with different data type.

“n = min(n1, n2, n3);” calls function template “min( )”, the “int” substitutes the “T”.

Class Template

One class template can substitute several classes' code.

If several class bodies are the same, only data types are different, you can use Class Template.

```
template <class T>      // define a class template
class  templateName{ class body };
```

The syntax to create class template object looks like this:

```
templateName  <dataType>object;
```

Example b5

```
#include <iostream>
using namespace std;
template< class T>    // define a class template
class Contrast{   //class template
public:
Contrast(T a, T b)  //constructor
{ x = a; y = b;}
T max( )    // “T” may be a different data type.
{ return (x > y) ? x : y;}
private:
T x, y;    // “T” may be a different data type.
};
int main( ){
Contrast  <int>con1( 10, 20);  // create an object “con1”
cout << con1.max( ) << endl;   // data type is “int”
Contrast  <float>con2( 20.01, 20.02 );  // create “con2”
cout << con2.max( ) << endl;   // data type is “float”
Contrast  <long> con3( 67891, 67892 );  // create “con3”
cout << con3.max( ) << endl;  // data type is “long”
return 0;
}
```

Output:

```
20
20.02
67892
```

Explanation:

“template< class T> class Contrast” defines a class template “Contrast”.

“Contrast <int>con1(10, 20);” creates an object “con1” and use class template “Contrast”.

Macros Definition

(1)

```
#define   identifier   string
```

“#define identifier string” is a macros definition, which is used to substitute “string” with the “identifier”.

Example b6

#define PI 3.1415926

Explanation:

Define “PI” as 3.1415926, therefore, “PI” can substitute 3.1415926.

(2)

```
#define   macrosName(parameters)   expression
```

“#define macrosName(parameters) expression” is a macros definition, which is used to substitutes “expression” with “macrosName(parameters)”.

Example b7

```
#include <iostream>
using namespace std;
#define   Multiply(x,y)   x*y    // macro definition
int main( ){
double area = Multiply(10, 20);  // Namely area=x*y;
cout << area << endl;
}
```

Output:

200

Explanation:

“#define Multiply(x,y) x*y” is a macro definition. It defines “Multiply(x,y) as x*y”. Therefore, “Multiply(x,y) can substitute x*y.

Conditional Compiling (1)

Part of the C++ program will be compiled if the condition test is true, it will not be compiled if the condition test is false.

```
#if expression
  compile code1       //if true, compile this.
#else
  compile code2       //if false, compile this.
#endif
```

Example b8

```
#include<iostream>
using namespace std;
#define ID 100
int main( ){
#if ID > 0
cout<< “ID>0” << endl;
#else
cout<< “ID<0” << endl;
#endif
return 0;
}
```

Output:

```
ID>0
```

Explanation:

Because ID>0, only the command “cout<< “ID>0” <<endl;”can be compiled.

Conditional Compiling (2)

Part of the C++ program will be compiled if the string has been defined previously, it will not be compiled if the string has not been defined.

```
#ifdef  identifier
  compile code1      //if defined, compile this.
#else
  compile code2      //if not defined, compile this.
#endif
```

Example b9

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

#define VARIABLE  “This is a string.”
int main( ){
#ifdef VARIABLE
cout<< “VARIABLE has been defined” <<endl;
#else
cout<< “VARIABLE has not been defined” << endl;
#endif
return 0;
}
```

Output:

VARIABLE has been defined.

Explanation:

Because “VARIABLE” has been defined previously, the command “cout<< “VARIABLE has been defined” <<endl;” can be compiled.

Exercises

Inline Function

Write C++ codes to your favorite editor.

```
#include <iostream>
using namespace std;
inline int sum(int n);  //define an inline function
int main( ){
int number;
cout << “Please enter a number: “;
cin >> number;
number = sum(number);
cout << “Result: ” << number << endl;
number = sum(number);
cout << “Result: ” << number << endl;
number = sum(number);
cout << “Result: ” << number << endl;
}
int sum(int n){
return n + n;
}
```

Save, Compile and Run the Program

Output:

Please enter a number**:** 3

Result**:** 6

Result**:** 12

Result**:** 24

Exercises

Macro Definition

Write C++ codes to your favorite editor.

```
#include <iostream>
using namespace std;
#define   Multiply(x,y)   x*y    // macro definition
int main( ){
double area = Multiply(10, 20);  // Namely area=x*y;
cout << area << endl;
}
```

Save, Compile and Run the Program

Output:

200

How time flies! It is time to say good-bye.

Ray Yao's books:

Linux Command Line (eBook)

Linux Command Line (Paper Book)

JAVA 100 Tests, Answers & Explanations

PHP 100 Tests, Answers & Explanations

JavaScript 100 Tests, Answers & Explanations

JAVA in 8 Hours

PHP in 8 Hours

JavaScript in 8 Hours

C++ in 8 Hours

AngularJS in 8 Hours

JQuery in 8 Hours

JavaScript 50 Useful Programs

Source Code for Download

Please download the source code of this book:

Source Code of C++ in 8 Hours

Dear My Friend,

If you are not satisfied with this eBook, could you please return the eBook for a refund within seven days of the date of purchase?

If you found any errors in this book, could you please send an email to me?

yaowining@yahoo.com

I will appreciate your email. Thank you very much!

Sincerely

Ray Yao

My friend, See you!