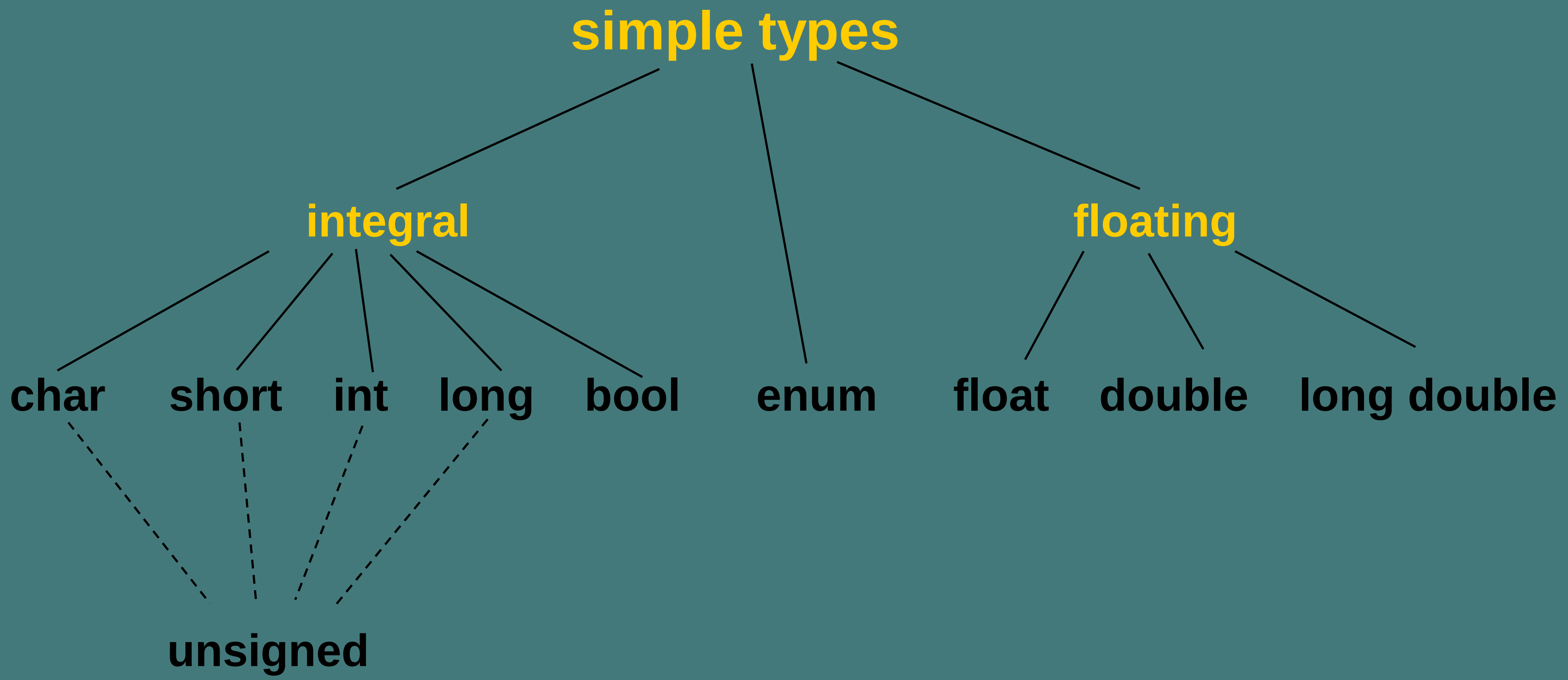
Chapter 3 Topics

- Constants of Type `int` and `float`
- Evaluating Arithmetic Expressions
- Implicit Type Coercion and Explicit Type Conversion
- Calling a Value-Returning Function
- Using Function Arguments
- Using C++ Library Functions in Expressions
- Calling a Void Function
- C++ Manipulators to Format Output
- String Operations `length`, `find`, `substr`

C++ Simple Data Types



simple types

integral

floating

char short int long bool

enum

float double long double

unsigned

Samples of C++ Data Values

int sample values

4578 -4578 0

float sample values

95.274 95. .265

9521E-3 -95E-1 95.213E2

char sample values

‘B’ ‘d’ ‘4’ ‘?’ ‘*’

Scientific Notation

2.7E4 means 2.7×10^{4} =

2.7000 =

27000.0

2.7E-4 means 2.7×10^{-4} =

0002.7 =

0.00027

What is an Expression in C++?

- An expression is a valid arrangement of variables, constants, and operators.
- in C++ each expression can be evaluated to compute a value of a given type
- the value of the expression

  9.3 * 4.5   is   41.85

Operators can be

binary	involving 2 operands	2 + 3
unary	involving 1 operand	-3 j++
ternary	involving 3 operands	*later*

Some C++ Operators

Precedence	Operator	Description
Higher	( ),++, --	Function call, Postfix op
	+, -	Positive, Negative
	++,--	Prefix op
	*	Multiplication
	/	Division
	%	Modulus (remainder)
	+	Addition
	-	Subtraction
Lower	=	Assignment

Precedence

- higher Precedence determines which operator is applied first in an expression having several operators

Division Operator

- the result of the division operator depends on the type of its operands
- if one or both operands has a floating point type, the result is a floating point type. Otherwise, the result is an integer type
- Examples

11 / 4	has value	2
11.0 / 4.0	has value	2.75
11 / 4.0	has value	2.75

Modulus Operator

- the modulus operator % can only be used with integer type operands and always has an integer type result
- its result is the integer type remainder of an integer division

EXAMPLE

11 % 4 has value 3 because

R = ?

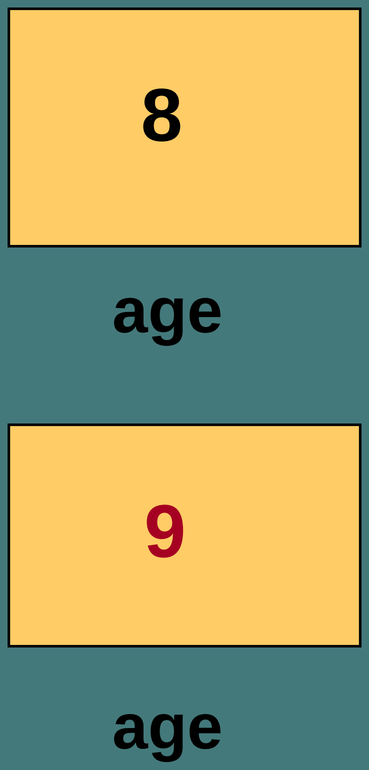
4) 11

More C++ Operators

```
int  age;


age = 8;


age = age + 1;
```

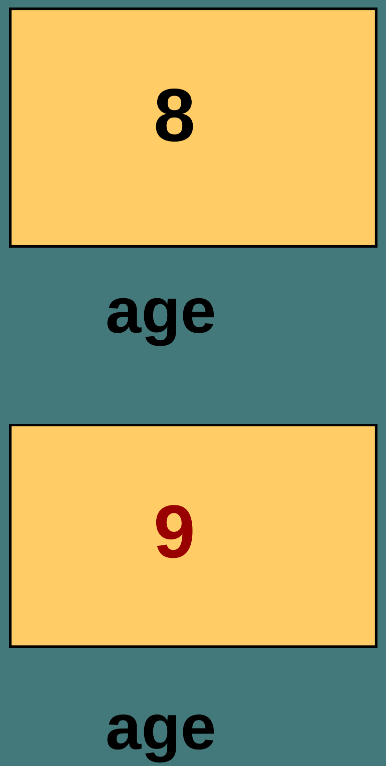


PREFIX FORM(前置形式) Increment Operator

```
int  age;

age = 8;

++age;
```



8

age

9

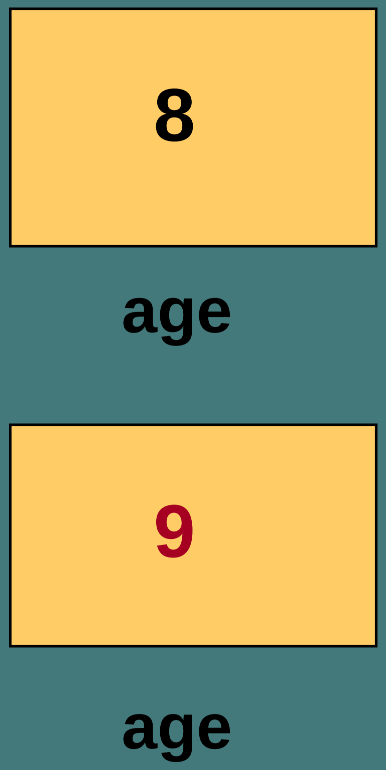
age

POSTFIX FORM (後置形式)
Increment Operator

```
int  age;

age = 8;

age++;
```



8

age

9

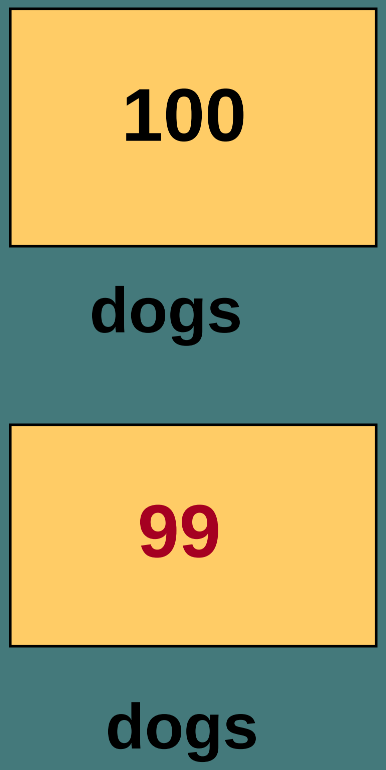
age

Decrement Operator

```
int  dogs;

dogs = 100;

dogs--;
```



100

dogs

99

dogs

Which Form to Use

- when the increment (or decrement) operator is used in a “*stand alone*” statement solely to add one (or subtract one) from a variable’s value, it can be used in either prefix or postfix form

USE EITHER

dogs-- ;

--dogs ;

(dogs ← dogs-1)

BUT...

- when the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield ***different*** results

- j + k++
- ++k +j //請注意這兩個運算式之區別

WE'LL SEE HOW LATER . . .

```
//This program tests for prefix/postfix operator ++
//
#include <iostream>
using namespace std ;
void main()
{    int j=9 , r1, r2 ;      //宣告變數
     const int c=100 ;
      r1= j++ + c ;
     cout << "the result of j++ +c is : " <<r1 << " and " <<" j is "<< j<<endl ;
      r2= ++j + c ;
      cout << "---------------------------------------- " << '\n';
     cout << "the result of ++j +c is : " <<r2 << " and " <<" j is "<< j<<endl ;
     system ("pause");
}
```

執行結果:

```
the result of j++ +c is : 109 and  j is 10
-----------------------------------------
the result of ++j +c is : 111 and  j is 11
請按任意鍵繼續 . . .
```

Assignment Operator Syntax

```
Variable  =  Expression
```

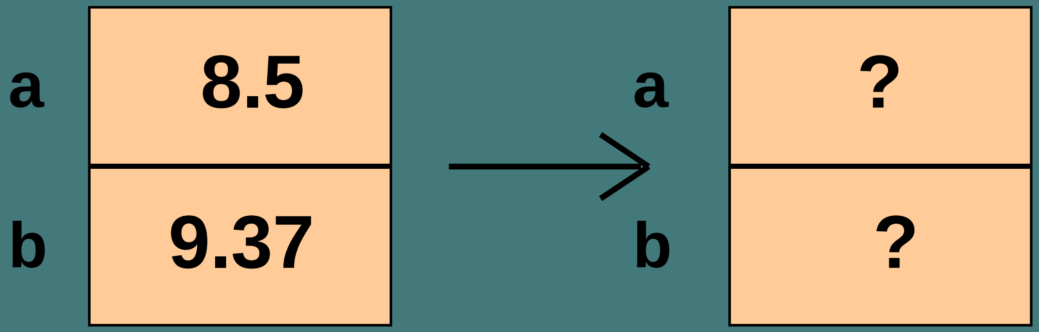
- first, Expression on right is evaluated
- then the resulting value is stored in the memory location of Variable on left

NOTE: An automatic type coercion occurs after evaluation but before the value is stored if the types differ for Expression and Variable

What value is stored?

```
float  a;
float  b;


a = 8.5;
b = 9.37;
a = b;
```



a	8.5
b	9.37

→

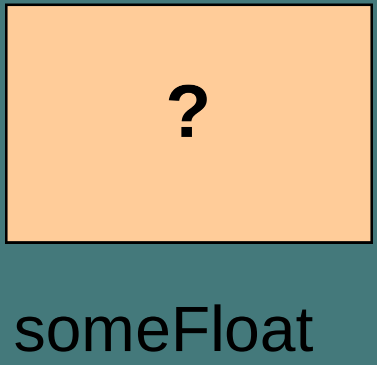
a	?
b	?

What is stored?

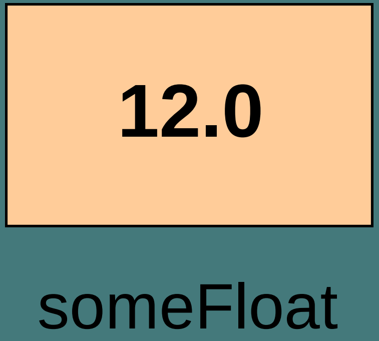
```
float  someFloat;



someFloat = 12;
```



// causes implicit type conversion



What is stored?

```
int  someInt;


someInt = 4.8;
```

?

someInt

// causes implicit type conversion

4

someInt

Type Casting is Explicit Conversion of Type

int(4.8)	has value	4
float(5)	has value	5.0
float(7/4)	has value	1.0
float(7) / float(4)	has value	1.75

Some Expressions

int age;

EXAMPLE	VALUE
age = 8	8
- age	- 8
5 + 8	13
5 / 8	0
6.0 / 5.0	1.2
float (4 / 8)	0.0
float (4) / 8	0.5
cout << “How old are you?”	cout
cin >> age	cin
cout << age	cout

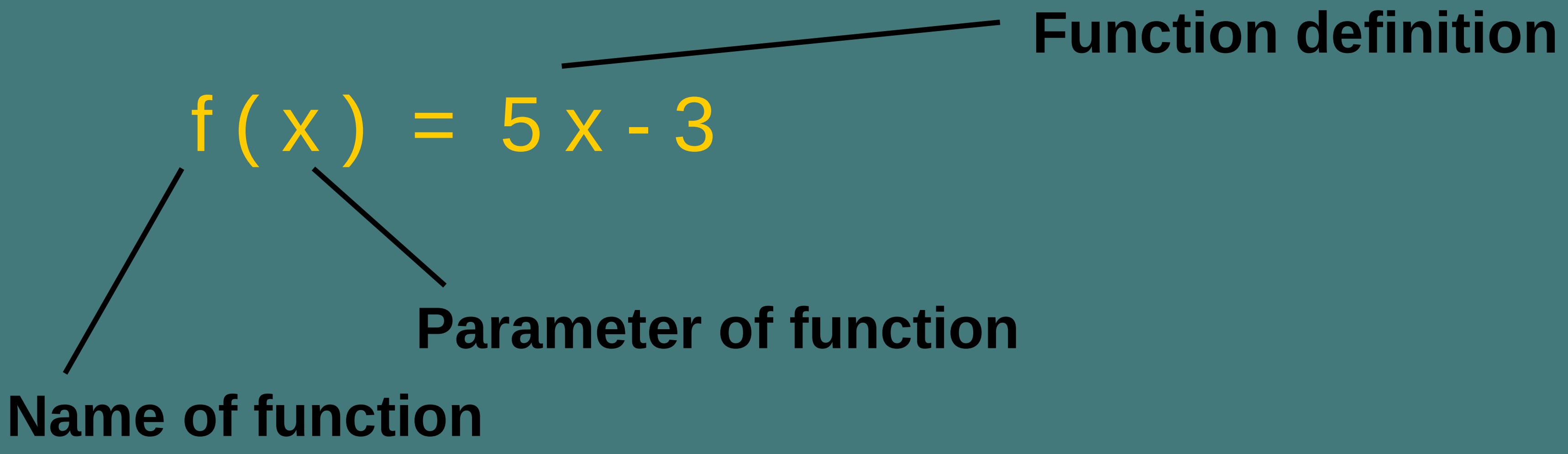
Functions

- every C program must have a function called main

- program execution always begins with function main

- any other functions are subprograms and must be called

Function Concept in Math



$$f(x) = 5x - 3$$

Function definition

Parameter of function

Name of function

When $x = 1$, $f(x) = 2$ is the returned value.

When $x = 4$, $f(x) = 17$ is the returned value.

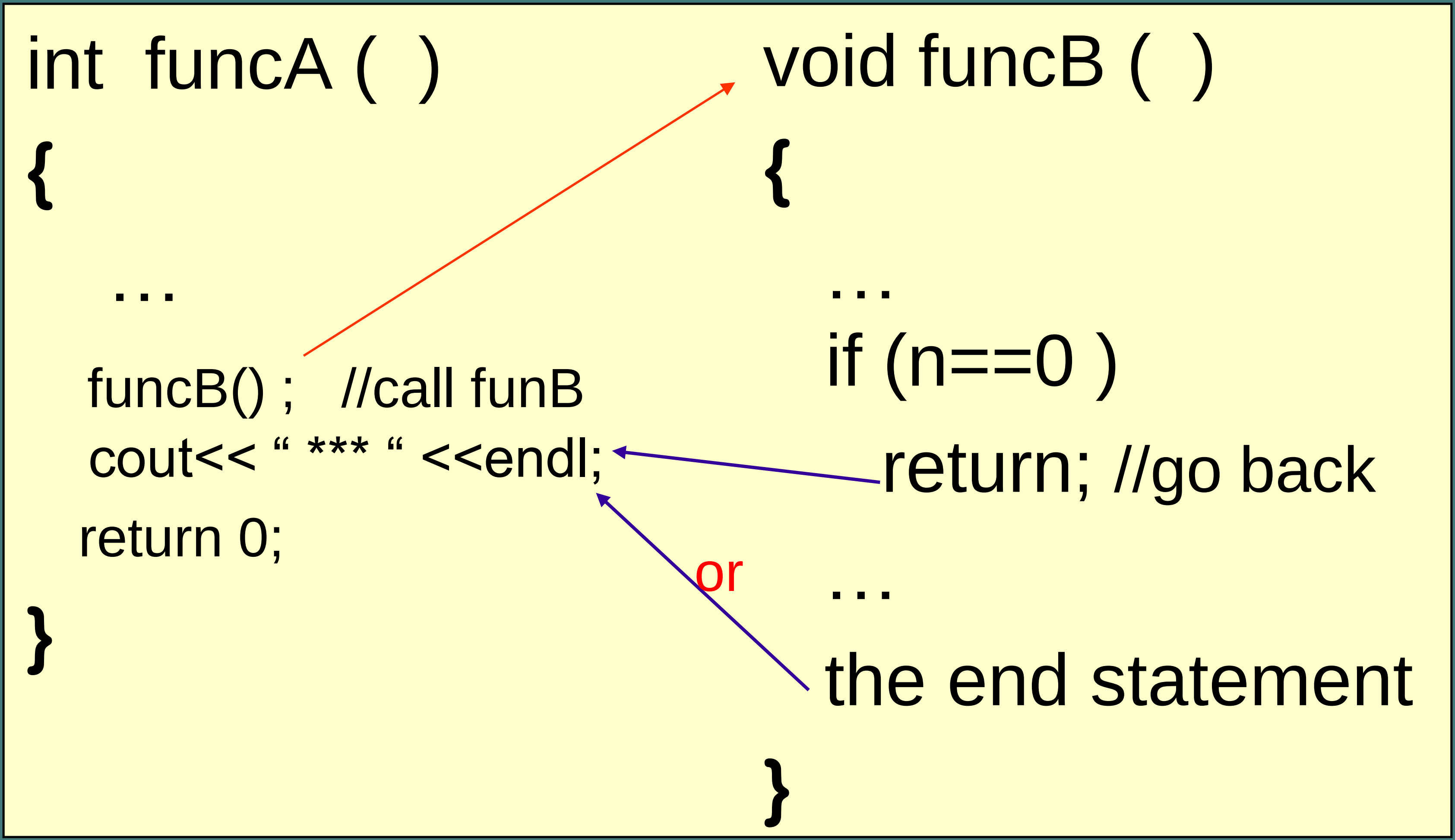
Returned value is determined by the function definition and by the values of any parameters.

Function Calls

- one function calls another by using the name of the called function together with ( ) containing an argument list

- a function call temporarily transfers control from the calling function to the called function. Once “return” meets or exits the called function, the control is going back to the caller.

Control flows between functions



```
int  funcA (  )
{
     …
   funcB() ;   //call funB
   cout<< “ *** “ <<endl;
   return 0;
}
```

```
void funcB (  )
{
     …
   if (n==0 )
      return; //go back
     …
   the end statement
}
```

More About Functions

- it is not considered good practice for the body block of function main to be long
- function calls are used to do tasks
- every C++ function has a return type
- if the return type is not void, the function returns a value to the calling block

Where are functions?

located in libraries

OR

written by programmers

HEADER	FUNCTION	EXAMPLE	VALUE
FILE		OF CALL	
<cstdlib>	abs(i)	abs(-6)	6
<cmath>	pow(x,y)	pow(2.0,3.0)	8.0
	fabs(x)	fabs(-6.4)	6.4
<cmath>	sqrt(x)	sqrt(100.0)	10.0
	sqrt(x)	sqrt(2.0)	1.41421
<cmath>	log(x) (natural)	log(2.0)	.693147
<iomanip>	setprecision(n)	setprecision(3)	

練習

- 1. include cmath library
  include iostream library
- 2. in main()
  declare a const variable of value 96.
  print out the value of square root of the variable .
  print out the value of logarithm of the variable .

練習 (續)

- 1. 同樣 include iostream library

  include cmath library

- 2. in main()

  declare a variable of floating-point number(浮點數)

  read in data from keyboard for the variable

  print out the value of square root of the variable .

  print out the value of logarithm of the variable .

Function Call Syntax

```
FunctionName (  Argument List  )
```

The argument list is a way for functions to communicate with each other by passing information.

The argument list can contain 0, 1, or more arguments, separated by commas, depending on the function.

A void function call stands alone

```
#include <iostream>

void  DisplayMessage ( int ) ;              // declares function
                                            //called by value

int main(  )
{
    DisplayMessage( 15 ) ;                  //function call

    cout  <<  “Good Bye“  <<    endl ;

    return 0 ;
}
```

A void function does NOT return a value

```
                                    // header and body here

void  DisplayMessage ( int  n )
{                                                   n  15
      cout  <<  "I have liked math for  "
                <<  n  <<  " years"  << endl ;
}
```

Two Kinds of Functions

Value-Returning	Void
Always returns a single value to its caller and is called from within an expression.	Never returns a value to its caller, and is called as a separate statement.

<< is a binary operator

<< is called the output or insertion operator

<< is left associative

EXPRESSION	HAS VALUE
cout << age	cout

STATEMENT

```
cout << “You are “  << age  <<  “ years old\n” ;
```

<iostream> is header file

- for a library that defines 3 objects

  an istream object named cin (keyboard)

  an ostream object named cout (screen)

  an ostream object named cerr (screen)

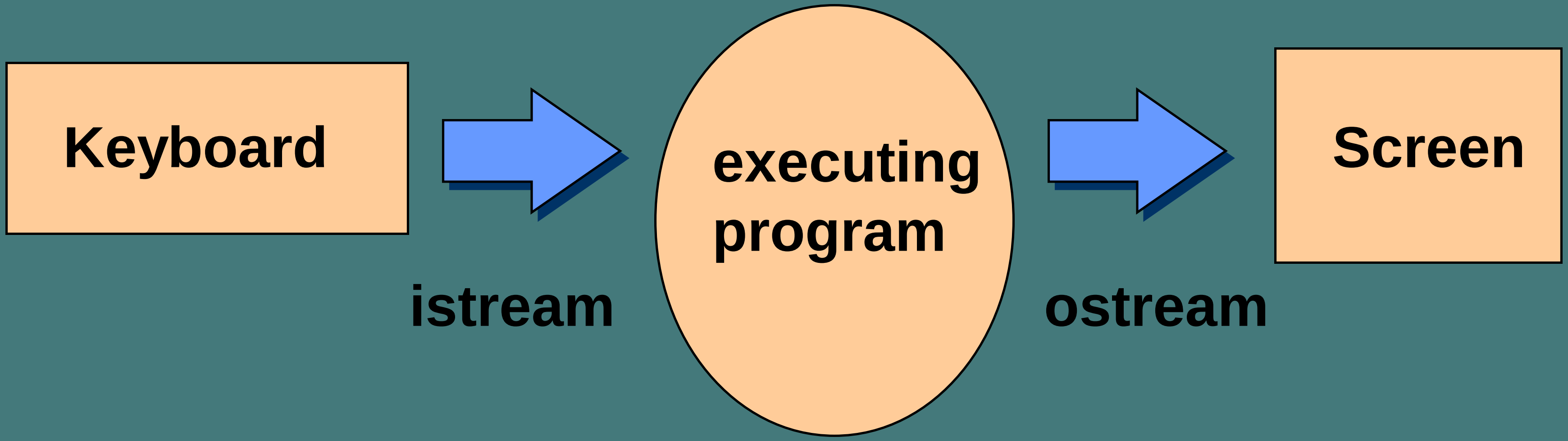
- istream cin ; //定義cin 是istream的物件

  ostream cout ;

  ostream cerr ; //輸出錯誤訊息之用

No I/O is built into C++

- instead, a library provides input stream and output stream



<iostream> is header file

- Declares all of its identifiers to be in a namespace called std :

```
namespace std
{ …
      Declarations of variables, data
      types, and so forth
 }
```

<iostream> is header file

-
  ```
  #include  <iostream>
  int main()
  {
     std::cout<<“happy day!!”<<std::endl ;
     return 0 ;
   }
  ```

Qualified name

-
  ```
  using namespace std ;
   …
   cout<<“happy day!!”<<endl ;
  ```

Manipulators

- manipulators are used only in input and output statements
- endl, fixed, showpoint, setw, and setprecision are manipulators that can be used to control output format
- endl is use to terminate the current output line, and create blank lines in output

Insertion Operator (<<)

- the insertion operator << takes 2 operands
- the left operand is a stream expression, such as cout
- the right operand is an expression of simple type, or a string, or a manipulator

Output Statements

SYNTAX (revised)

```
cout  <<  ExpressionOrManipulator
          <<  ExpressionOrManipulator  . . . ;
```

Output Statements

SYNTAX

```
cout  <<  Expression   << Expression  . . . ;
```

These examples yield the same output.

```
cout  <<  “The answer is “ ;
cout  <<  3 * 4 ;
```

```
cout  <<  “The answer is “  <<  3 * 4 ;
```

Using Manipulators Fixed and Showpoint

- use the following statement to specify that (for output sent to the cout stream) **decimal format** (not scientific notation) be used, and that a **decimal point** be included (even for floating values with 0 as fractional part)

```
cout  <<   fixed  <<   showpoint ;
```

setprecision(n)

- requires #include <iomanip> and appears in an expression using insertion operator (<<)
- if fixed has already been specified, argument n determines the number of places displayed after the decimal point for floating point values
- remains in effect until explicitly changed by another call to setprecision

What is exact output?

```
#include  <iomanip>                      // for setw( ) and setprecision( )
#include  <iostream>

using  namespace  std;

int main (  )
{
   float   myNumber  =  123.4587 ;

   cout  <<  fixed  <<   showpoint  ;       // use decimal format
                                            // print decimal points

   cout  <<  “Number is ”  <<  setprecision ( 3 )
           <<  myNumber     <<  endl ;

   return  0 ;
}
```

OUTPUT

```
Number is 123.459
```

value is **rounded** if necessary to be displayed with exactly **3 places** after the decimal point

Manipulator setw

- “set width” lets us control how many character positions the next data item should occupy when it is output

- setw is only for formatting numbers and strings, not char type data

setw(n)

- requires #include <iomanip> and appears in an expression using insertion operator (<<)

- argument n is called the fieldwidth specification, and determines the number of character positions in which to display a right-justified number or string (not char data). The number of positions used is expanded if n is too narrow

- “set width” affects only the very next item displayed, and is useful to align columns of output

What is exact output?

```
#include  <iomanip>                              // for setw( )
#include  <iostream>
#include  <string>

using  namespace  std;

int  main (  )
{
   int  myNumber    =  123 ;
   int  yourNumber  =  5 ;

   cout  <<  setw ( 10 )    <<   “Mine”
          <<   setw ( 10 )    <<  “Yours”          <<   endl;
          <<   setw ( 10 )    <<  myNumber
          <<   setw ( 10 )    <<  yourNumber   <<   endl ;

   return 0 ;
}
```

OUTPUT

```
position    12345678901234567890
                  Mine     Yours
                   123         5
```

each is displayed right-justified and each is located in a total of 10 positions

What is exact output?

```
#include  <iomanip>                        // for setw( ) and setprecision( )
#include  <iostream>

using  namespace  std;

int  main ( )
{
   float myNumber    =  123.4 ;
   float yourNumber  =  3.14159 ;

   cout  <<  fixed  <<   showpoint  ;       // use decimal format
                                            // print decimal points

   cout  <<  “Numbers are: ”  <<  setprecision ( 4 ) <<  endl
         <<   setw ( 10 )         <<  myNumber      <<   endl
         <<   setw ( 10 )         <<  yourNumber   <<   endl ;

   return 0 ;
}
```

OUTPUT

```
12345678901234567890
Numbers are:
  123.4000
    3.1416
```

each is displayed right-justified and rounded if necessary and each is located in a total of 10 positions with 4 places after the decimal point

More Examples

312.0

x

4.827

y

```
float  x   =  312.0 ;
float  y   =  4.827 ;

cout  <<  fixed  <<   showpoint  ;

cout  <<  setprecision ( 2 )
      <<   setw ( 10 )      <<  x   <<   endl
      <<   setw ( 10 )      <<  y   <<   endl ;

cout  <<  setprecision ( 1 )
      <<   setw ( 10 )      <<  x   <<   endl
      <<   setw ( 10 )      <<  y   <<   endl ;

cout  <<  setprecision ( 5 )
      <<   setw ( 7 )     <<  x   <<   endl
      <<   setw ( 7 )     <<  y   <<   endl ;
```

OUTPUT

```
''''312.00
''''''4.83


'''''312.0
'''''''4.8


312.00000
4.82700
```

HEADER FILE	MANIPULATOR	ARGUMENT TYPE	EFFECT
<iostream>	endl	none	terminates output line
<iostream>	showpoint	none	displays decimal point
<iostream>	fixed	none	suppresses scientific notation
<iomanip>	setw(n)	int	sets fieldwidth to n positions
<iomanip>	setprecision(n)	int	sets precision to n digits

利用定點格式顯示一個資料表

```
#include <iostream>
#include  <iomanip>
#include <cmath>
using namespace std ;
void main()
{
    float num=0;                               //declare num is a real number
    const float step = 0.2;            //declare step is a step-size
    //to create a table to list the square root the numbers 0: 0.2: 2
    //with fixed precision
   cout << fixed << showpoint ;      // use decimal format
   cout << “  Numbers are : ” << “     ” << “Square roots are: ” <<
    ‘\n’ <<   endl ;                             //先列印資料表的表頭
int j = 0 ;

while (j <= 10)
{
    cout << setprecision(2) << setw(10) << num<< "          " ;
    cout << setprecision(4) << setw(10) << sqrt(num) <<
    endl ;

    j++;
    num = j*step ;
}
    system("pause") ;  //to keep the screen output
}
```

```
Numbers are :      Square roots are:

        0.00            0.0000
        0.20            0.4472
        0.40            0.6325
        0.60            0.7746
        0.80            0.8944
        1.00            1.0000
        1.20            1.0954
        1.40            1.1832
        1.60            1.2649
        1.80            1.3416
        2.00            1.4142
請按任意鍵繼續 . . .
```

Using namespace

- header file – iostream
  declares all of its identifiers to be in a namespace called std :
  { …
  Declarations of variables, data types, and so forth
  }
- qualified name : std::cout
- “:: “ -- scope resolution operator

String

- C++ 提供的一個類別(class):文字串
- string operation ：

  通常是類別內定義的公用函數(public member)

  size() length() find( ) substr( )
- string firstName ; //宣告firstName是string物件

  string::size_type len ; //宣告len的資料型態

  firstName = ”Alexandra” ;

  len = firstName.length() ;

`length` Function

- function `length` returns an unsigned integer value that equals the number of characters currently in the string
- function `size` returns the same value as function length
- you must use dot notation ‘.’ in the call to function `length` or `size`

find Function

- function find returns an unsigned integer value that is the beginning position for the first occurrence of a particular substring within the string

- the substring argument can be a string constant, a string expression, or a char value

- if the substring was not found, function find returns the special value string::npos
“not a position within the string”

substr Function

- function substr returns a particular substring of a string
- the first argument is an unsigned integer that specifies a starting position within the string
- the second argument is an unsigned integer that specifies the length of the desired substring
- positions of characters within a string are numbered starting from 0, not from 1

What is exact output?

```
#include  <iostream>
#include  <string>                    // for functions length, find, substr

using  namespace  std;

int  main (  )
{
   string  stateName = “Mississippi” ;

   cout  <<  stateName.length( )  << endl;

   cout  <<   stateName.find(“is”) <<  endl;

   cout <<  stateName.substr( 0, 4 ) <<  endl;

   cout << stateName.substr( 4, 2 ) <<  endl;

   cout << stateName.substr( 9, 5 ) <<  endl;

   return 0 ;
}
```

What is exact output?

```
#include  <iostream>
#include  <string>                        // for functions length, find, substr

using  namespace  std;

int  main ( )
{
   string  stateName = “Mississippi” ;

   cout  <<  stateName.length( )  << endl;              // value 11

   cout  <<   stateName.find(“is”) <<  endl;           // value 1

   cout <<  stateName.substr( 0, 4 ) <<  endl;        // value “Miss”

   cout << stateName.substr( 4, 2 ) <<  endl;        // value “is”

   cout << stateName.substr( 9, 5 ) <<  endl;        // value “pi”

   return 0 ;
}
```

What is exact output?

- **cout << stateName.find(“is”) << endl;**

  如果在上面敘述中加入下列敘述

- **cout << stateName.find(“os”) << endl;**

  則輸出如下：

```
11
1
4294967295
Miss
is
pi
```

string::npos

runprogam

Giving a Value to a Variable

In your program you can assign (give) a value to the variable by using the **assignment operator =**

```
ageOfDog = 12;
```

or by another method, such as

```
cout << “How old is your dog?”;
cin  >> ageOfDog;
```

>> is a binary operator

>> is called the input or extraction operator

>> is left associative

EXPRESSION	HAS VALUE
cin >> age	cin

STATEMENT

```
cin  >>  age  >>  weight ;
```

Extraction Operator (>>)

- variable cin is predefined to denote an input stream from the standard input device (the keyboard)
- the extraction operator >> called “get from” takes 2 operands. The left operand is a stream expression, such as cin , the right operand is a variable of simple type.
- operator >> attempts to extract the next item from the input stream and store its value in the right operand variable

Input Statements

SYNTAX

cin >> *Variable* >> *Variable* . . . ;

These examples yield the same result.

```
cin  >>  length ;
cin  >>  width ;
```

```
cin  >> length  >> width ;
```

Extraction Operator >>

“skips over”

(actually reads but does not store anywhere)

leading white space characters

as it reads your data from the input stream (either keyboard or disk file)

Whitespace Characters Include . . .

- blanks
- tabs
- end-of-line (newline) characters

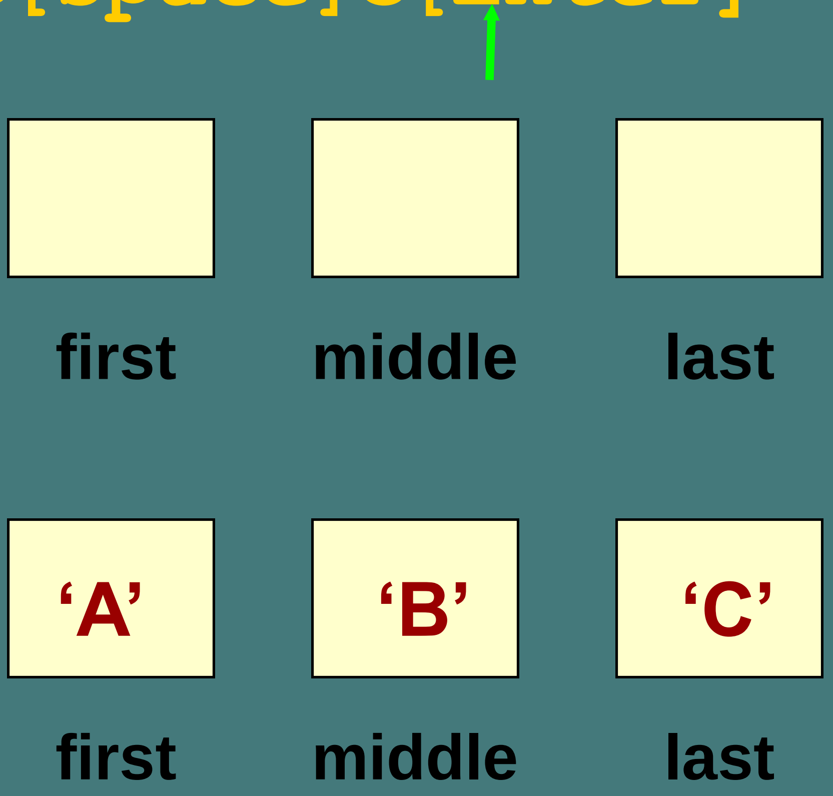
The newline character is created by hitting Enter or Return at the keyboard, or by using the manipulator endl or '\n' in a program.

At keyboard you type:

```
A[space]B[space]C[Enter]
```

```
char   first ;
char   middle ;
char   last ;
```

```
cin  >>   first  ;
cin  >>   middle  ;
cin  >>   last  ;
```



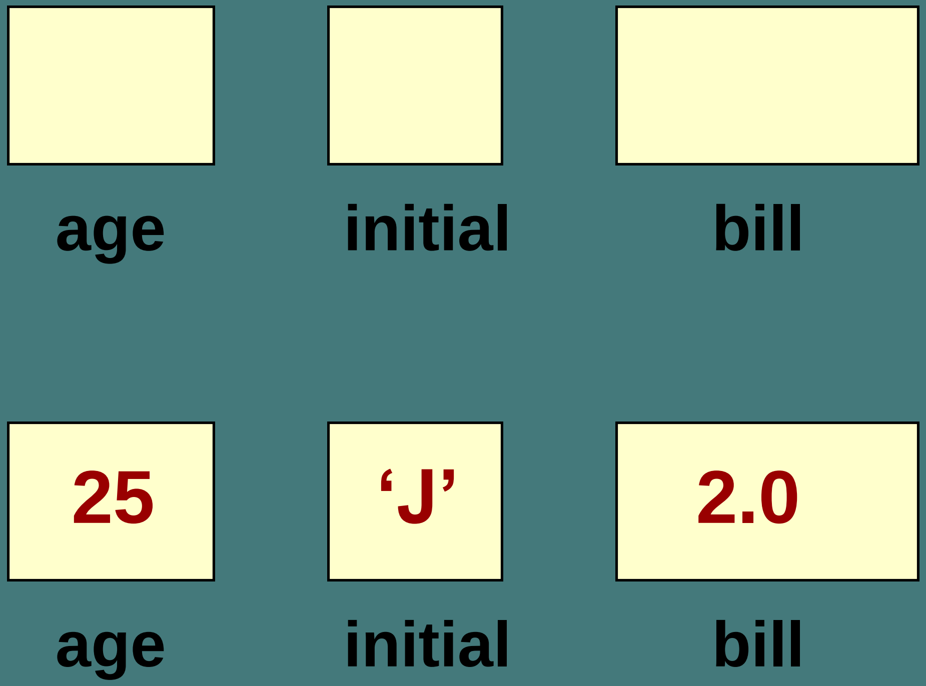
NOTE: A file reading marker is left pointing to the newline character after the 'C' in the input stream.

At keyboard you type:

```
[space]25[space]J[space]2[Enter]
```

```
int     age ;
char    initial ;
float   bill ;
```

```
cin  >>  age ;
cin  >>  initial ;
cin  >> bill ;
```



NOTE: A file reading marker is left pointing to the newline character after the 2 in the input stream.

Another example using >>

NOTE: □ shows the location of the file reading marker

STATEMENTS	CONTENTS			MARKER POSITION
int i ; char ch ; float x ;	□ i	□ ch	□ x	25 A\n 16.9\n
cin >> i ;	25 i	□ ch	□ x	25 A\n 16.9\n
cin >> ch ;	25 i	‘A’ ch	□ x	25 A\n 16.9\n
cin >> x ;	25 i	‘A’ ch	16.9 x	25 A\n 16.9\n

String Input in C++

Input of a string is possible using the extraction operator >>.

(文字串類別也可以使用>>輸入資料)

EXAMPLE

```
string    message ;
cin    >>  message ;
cout  <<  message ;
```

HOWEVER . . .

Extraction operator >>

When using the extraction operator (>>) to read input characters into a string variable:

- the >> operator skips any leading whitespace characters such as blanks and newlines
- it then reads successive characters into the string, and stops at the first trailing whitespace character (which is not consumed, but remains waiting in the input stream)

String Input Using >>

```
string   firstName ;
string   lastName ;
cin  >>  firstName >> lastName ;
```

Suppose input stream looks like this:

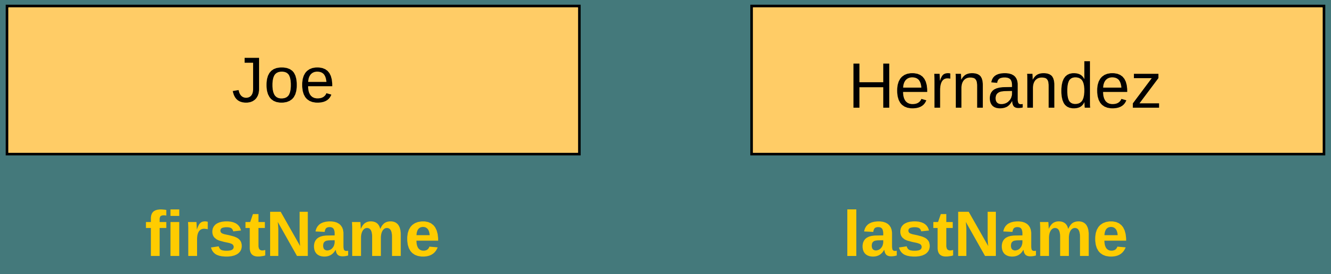
□□Joe□Hernandez□23

WHAT ARE THE STRING VALUES?

Results Using >>

```
string   firstName ;
string   lastName ;
cin  >>  firstName >> lastName ;
```

RESULT



Joe

Hernandez

firstName

lastName