C++ Programming Language
2020-2021
المرحلة الاولى / الفصل الاول / الفصل الثاني
أستاذ المادة

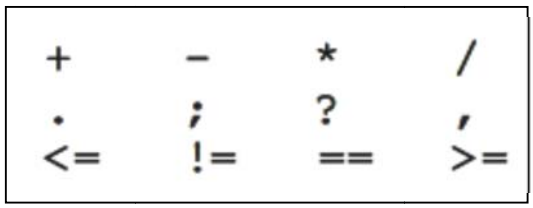
أ. م. د براء علي عطية
م.د.امنة داحم عبود

```
#include<iostream>

using namespace std;

intmain()
{
      i.ntnum;

      num … 6;

      cout<< "My     firstC++program."<< endl;
      cout  <<  "The  sumof2and3="<< 5<<endl;
      cout  <<  "7  +  8      "<< 7+8<<endl;
      cout<< "Num          "<< num<<endl;

      return 0;
}
```

Sample Run: (When you compile and execute this program, the following four lines are displayed on the screen.)

```
My first C++ program. The sum of 2 and 3 =5
7 + 8      15
Num =  6
```

Following are some of the special symbols:



Order of Precedence

When more than one arithmetic operator is used in an expression, C++ uses the operator precedence rules to evaluate the expression. According to the order of precedence rules for arithmetic operators,

*, *I*, %

are at a higher level of precedence than:

+ -

Note that the operators *, *I,* and % have the same level of precedence. Similarly, the operators + and - have the same level of precedence.

When operators have the same level of precedence, the operations are performed from left to right. To avoid confusion, you can use parentheses to group arithmetic expressions . For example, using the order of precedence rules,

3* 7-6+ 2* 5*I* 4+ 6

means the following:

(((3 * 7) - 6) + ((2 4)) + 6	
* 5) *I*	(Evaluate *)
< (21 - 6) + (10 *I* 4))	(Evaluate *I*. Note that this is an integer division.)
+ 6	(Evaluate -)
((21- 6) + 2) + 6	(Evaluate first +)
(15 + 2) + 6	(Evaluate +)
17+ 6	
23	

The following C++ program shows the effect of the preceding statements:

```
// This program illustrates how data in the variables are
// manipulated.

#include<iostream>
#include<string>

        using namespace std;

int main()
        {
            int numl, num2; double sale; char first; string str;

            numl     4;
            cout<< "numl= "<<numl<< end!;

            num2 = 4*  5 - 11;
            cout<< "num2 = "<< num2 << end!;

            sale= 0.02*  1000;
            cout<< "sale = "<< sale << end!;

            first= 'D';
            cout<< "first = "<< first << end!;

            str ="It is a sunny day.";
            cout<< "str          "<< str<< end!;

            return 0;
```

Sample Run:

```
        numl     4
        num2 = 9
sale = 20
first = D
str = It is a sunny day.
```

What is C++ ?

- A programming language.
- A general-purpose, high-level, Compiled. staticly typed programming language.

General purpose languages can be used for any application and are not domain specific. HTML is an example of a domain-specific language.

C++ is a general-purpose object-oriented programming language developed by Bjarne Stroustrup of Bell Labs in 1979.... It was renamed C++ in 1983, but retains a strong link to C, and will compile most C programs. Compared to
C, C++ added object-oriented features to C such as classes, abstraction, and inherita nee.

What is programming...

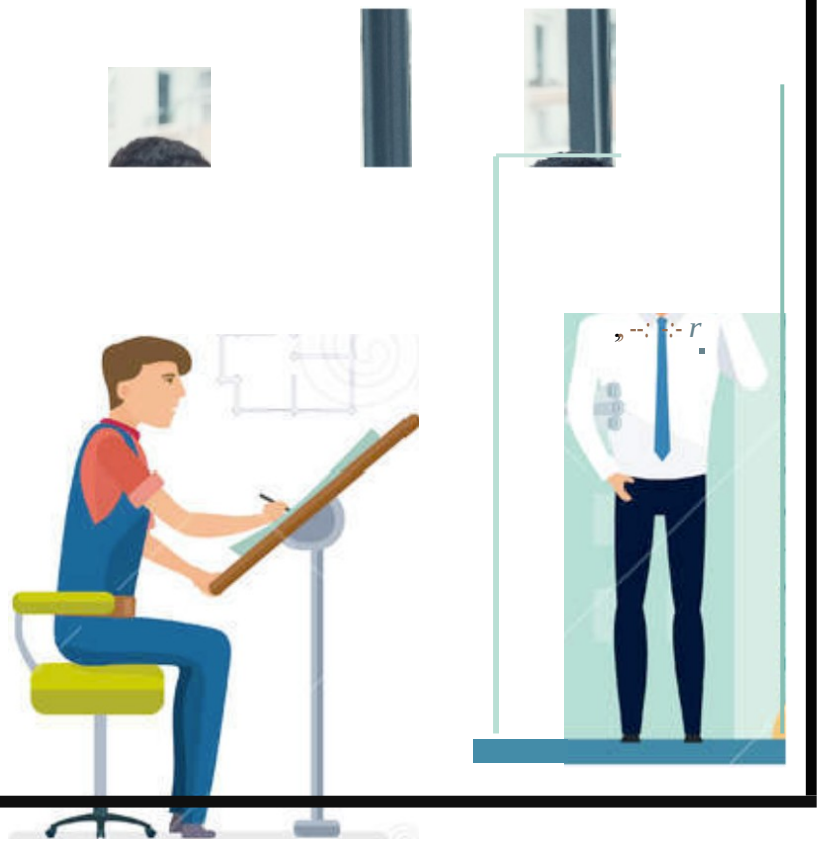
1. The process of writing computer programs.
2. Decomposing a problem and solving it with computer code.
3. A broad range of complementary skills across a framework of competencies.

What is programming as a profession?

- What portion do you think is related to programming?
- How much time do you think I spend programming?

- Most senior professionals spend very little time programming…

  …but their understanding of programming and its limitations is critical



Array (One-dimensional array or 1D array)

An array is a collection of a fixed number of components all of the same data type. A one-dimensional array is an array in which the components are arranged in a list form. Individual elements are referred to using common name and unique index of the elements. The array itself is given name and its elements are referred to by their subscripts. In C++, an array is denoted as follows:

The general form for declaring a one-dimensional array is:

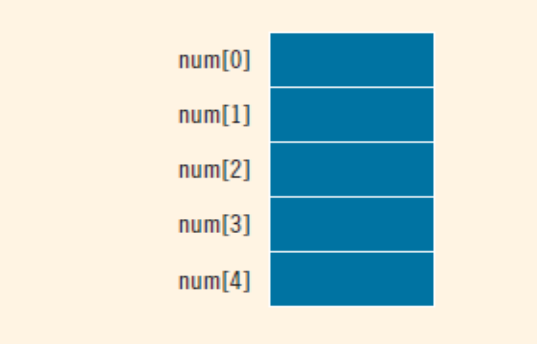
```
dataType arrayName[intExp];
```

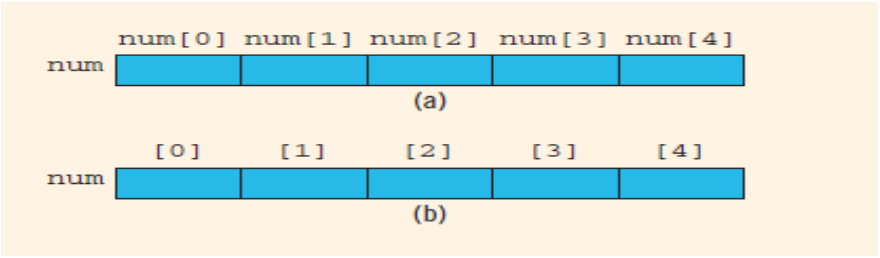
[ ] array subscriber int num[5+3];
int num[8];

in which intExp is any constant expression that evaluates to a positive integer. Also, intExp specifies the number of components in the array.

For example, the statement:
int num[5];

declares an array num of five components. Each component is of type int. The components are num[0], num[1], num[2], num[3], and num[4]. Figure below illustrates the array num.




num[0] num[1] num[2] num[3] num[4]
num
(a)
[0] [1] [2] [3] [4]
num
(b)

Accessing Array Components

The general form (syntax) used for accessing an array component is:

```
arrayName[indexExp]
```

in which indexExp, called the index, is any expression whose value is a nonnegative integer and should be ranged from 0 to intExp – 1 (i.e. from 0 to the number of components in the array minus one).

The index value specifies the position of the component in the array.

In C++, [ ] is an operator called the arraysubscriptingoperator. Moreover, in C++, the array index starts at 0.

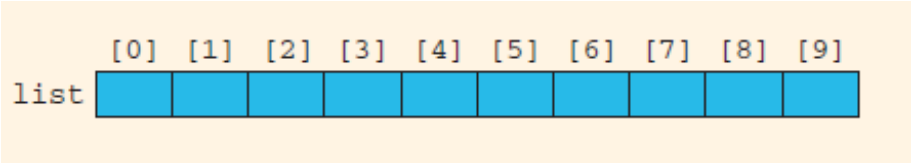
Consider the following statement:

```
int list[10];
```

This statement declares an array list of 10 components. The components are list[0], list[1], . . ., list[9]. In other words, we have declared 10 variables

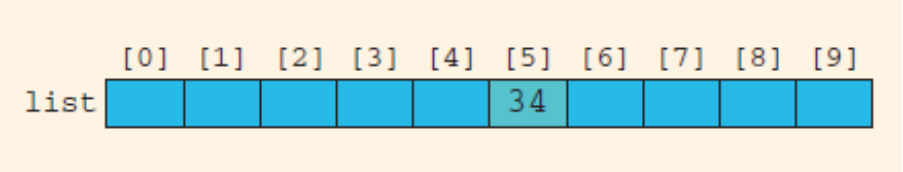
(as shown in the following figure).

```
int  list[10];
```



The assignment statement: list[5] = 34;

stores (i.e. assigns) 34 in list[5], which is the sixth component of the array list:

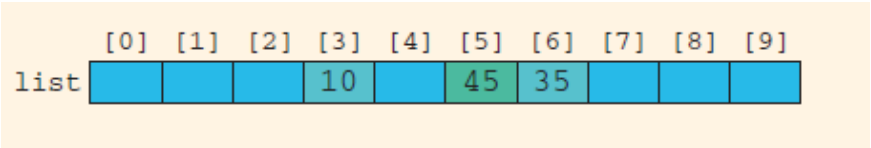


Suppose i is an int variable. Then, the assignment statement: list[3] = 63;
is equivalent to the assignment statements: i = 3;
list[i] = 63;
If i is 4, then the assignment statement:

int i = 4;
list[2 * i - 3] = 58; list[5] = 58;
stores 58 in list[5] because 2 * i - 3 evaluates to 5. The index expression is evaluated first, giving the position of the component in the array.

Next, consider the following statements: list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];

The first statement stores 10 in list[3], the second statement stores 35 in list[6], and the third statement adds the contents of list[3] and list[6] and stores the result in list[5] (see Figure below).



Processing One-Dimensional Arrays

Some of the basic operations performed on a one-dimensional array are initializing, inputting data, outputting data stored in an array, and finding the largest and/or smallest element. Moreover, if the data is numeric, some other basic operations are finding the sum and average of the elements of the array.

Each of these operations requires the ability to step through the elements of the array. This is easily accomplished using a loop. For example, suppose that we have the following statements:

```
int list[100]; //list is an array of size 100
int i;
```

The following for loop steps through each element of the array list, starting at the first element of list:

```
for (i = 0; i < 100; i++) //Line 1
//process list[i] //Line 2
```

```
main()
{
 int a[100]; cin >> a[0]; cin >>
          a[1]; cin >> a[2];
                    .
                    .
                    .
   cin >> a[99];
}
```

```
main()
{
  int a[100];
  for(int i = 0; i < 100; i++)
  {
       cout << "please, enter an integer number" << endl; cin >> a[i];
  }
}
```

```
main()
{
   int a[100];
   for(int i = 0; i < 100; i++) cin >> a[i];
 }

 main()
 {
   int a[100];
   for(int i = 0; i <= 99; i++) cin >> a[i];
 }

 main()
 {
   int a[100];
   for(int i = 99; i >= 0; i--) cin >> a[i];
 }
```

```
main()
{
```

```
        int a[100];
        for(int i = 0; i < 100; i++)
                cin >> a[i];
        for(int i = 0; i < 100; i++)
                cout << a[i] << endl;
        for(int i = 99; i >= 0; i--)
}               cout << a[i] << endl;
        for(int i = 0; i < 100; i= i+2)
                cout << a[i] << endl;
```

```
const int n = 100; int a[n];
for(int i = 0; i < n; i++)
        cin >> a[i]; for(int i = 0; i < n;
i++)
        cout << a[i] << endl; for(int i = n-1; i >= 0;
i--)
        cout << a[i] << endl; for(int i = 0; i < n; i=
i+2)
        cout << a[i] << endl; for(int i = 0; i < n; i=
i++)
    if(a[i]%2 == 1)
        cout << a[i] << endl;

int sum = 0;
for(int i = 0; i < n; i++)
    sum = sum + a[i]; cout << sum;
int sum_e = 0; int c_e = 0;
for(int i = 0; i < n; i++)
        if(a[i]%2 == 0){
            c_e++;
            sum_e = sum_e + a[i];
        }
cout << sum_e << endl;
cout << sum_e / c_e << endl; int max = a[0];
for(int i = 1; i < n; i++)
    if(a[i] > max)
        max = a[i]; cout << max <<
endl;
```

```
int max = -9999;
        for(int i = 0; i < n; i++)
            if(a[i]%2 == 0 && a[i] > max) max = a[i];
cout << max << endl;


}
```

If processing the list requires inputting data into list, the statement in Line 2 takes the form of an input statement, such as the cin statement. For example, the following statements read 100 numbers from the keyboard and store the numbers in list:

```
main()
{
        int list[100];
        int i;
        for (i = 0; i < 100; i++)
                cin >> list[i];
}
```

Initialize One Dimensional Array in C++

Here is an example, declaring and initializing values to the array name arr of type int, containing 10 elements

```
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Lecture of the Two dimensional arrays (Matrices) 2D Array =

Matrix

Two-dimensional array: A collection of a fixed number of components arranged in rows and columns (that is, in two dimensions), wherein all components are of the same type. The syntax for declaring a two-dimensional array is:

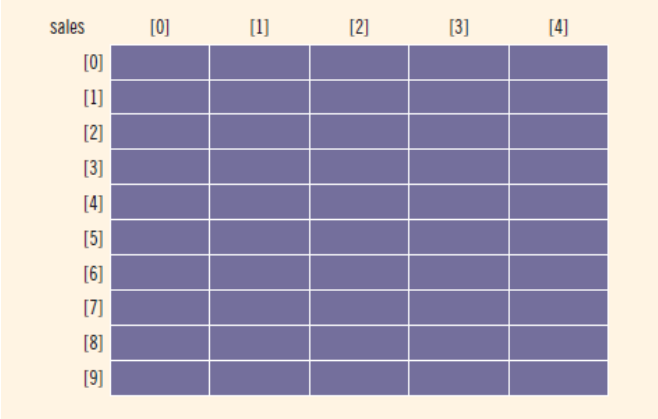
```
dataType  arrayName[intExp1][intExp2];
```

wherein intExp1 and intExp2 are constant expressions yielding positive integer values. The two expressions, intExp1 and intExp2, specify the number of rows and the number of columns, respectively, in the array.

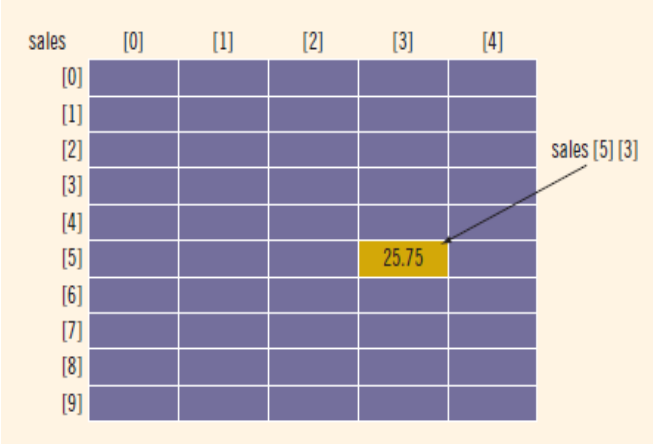
The statement:

double sales[10][5];

declares a two-dimensional array sales of 10 rows and 5 columns, in which every component is of type double. As in the case of a one-dimensional array, the rows are numbered 0. . .9 and the columns are numbered 0. . .4:



To access the components of a two-dimensional array, you need a pair of indices: one for the row position and one for the column position.



1) Two-Dimensional Array Initialization During Declaration

Like one-dimensional arrays, two-dimensional arrays can be initialized when they are declared. The following example helps illustrate this concept. Consider the following statement:

```
int board[4][3] = {{2, 3, 1},
                   {15, 25, 13},
                   {20, 4, 7},
                   {11, 18, 14}};
```

Initialization of the first row :]Comment [DS21

Initialization of the 2nd row :]Comment [DS22

Initialization of the 4th row :]Comment [DS23

This statement declares board to be a two-dimensional array of four rows and three columns. The components of the first row are 2, 3, and 1; the components of the second row are 15, 25, and 13; the components of the third row are 20, 4, and 7; and the components of the fourth row are 11, 18, and 14, respectively.

board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7
[3]	11	18	14

2) Two-Dimensional Array Assignment Through cin Function

For example, we have 3 students, each has 3 marks. Declare a 2D array and read the marks for each student using cin function.

```
main(){
        const int n = 3;
        int marks[n][n];
        int i, j;
        for(i=0; i < n; i++)
                for(j=0; j < n; j++){
                        cout<<"please, enter a mark for student" << i <<endl;
                        cin>> marks[i][j];
                } // end for j
} // end main
```

In memory:

i=0 1 2 3

j=0 1 2 3 0 1 2 3 0 1 2 3

marks

marks[0][0]=90	marks[0][1]=85	marks[0][2]=95
marks[1][0]=60	marks[1][1]=85	marks[1][2]=70
marks[2][0]=93	marks[2][1]=90	marks[2][2]=80

```
please, enter a mark for student  0
90
please, enter a mark for student  0
85
please, enter a mark for student  0
95
please, enter a mark for student  1
60
please, enter a mark for student  1
85
please, enter a mark for student  1
70
please, enter a mark for student  2
93
please, enter a mark for student  2
90
please, enter a mark for student  2
80
```

Outer loop for the :]Comment [DS24 rows of the matrix

Loop control :]Comment [DS25 variable

Inner loop for the :]Comment [DS26 columns of the matrix

You need a nested :]Comment [DS27 loop to visit all the rows and all the columns in the matrix.

LCV :]Comment [DS28

This LCV points to :]Comment [DS29 the rows of the matrix marks

This LCV points to :]Comment [DS210 the columns of the matrix marks

Questions:

Write a C++ program to read a 2D array of integer numbers. Then, find the following:

1. **The sum of all even elements.**
2. **The largest element in the matrix.**
3. **The smallest element in each row.**
4. **The sum of the smallest odd element in each column.**
5. **Swap between the elements of the main diagonal and the minor diagonal.**
6. **Print the elements of the left triangle of the main diagonal.**
7. **The sum of the elements of the right triangle of the minor diagonal.**
8. **Print even elements of odd rows.**
9. **Print odd elements located at the even columns.**
10. **Print the index of the largest even number in the matrix.**
11. **Copy the factorial of each element in a second matrix. What is the size of this matrix?**

```
main(){
const int n = 5;
int a[n][n];
int i,j;
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                cin>> a[i][j];
// Answer for (1)
int sum = 0;
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                if(a[i][j]%2 == 0)
                        sum = sum + a[i][j];// sum += a[i][j];
cout<<sum<<endl;
// Answer for (2)
int Large = -9999999; for(i=0;
i<n; i++){
        for(j=0; j<n; j++){
                if(a[i][j] > Large){
                        Large = a[i][j];
                        cout<< Large;
                }
                cout<< Large;
        }
        cout<< Large;
}
```

For reading the elements of the matrix. Visiting the elements row by row, and column followed by column. :]Comment [DS211

```
cout<< Large;
// Answer for (3) int
small;
for(i=0; i< n; i++){
        small = +9999999;
        for(j=0; j<n; j++)
                if(a[i][j] < small)
                        small = a[i][j];
        cout<< small << "in row"<< i << endl;
}
// Answer for (4)
int sum_odd = 0;
int small_odd;
for(i=0; i<n; i++){
        small_odd = +9999999;
        for(j=0; j<n; j++)
                if(a[j][i]%2 == 1 && a[j][i] < small_odd)
                        small_odd = a[j][i];
        sum_odd += small_odd;
}

cout<<sum_odd;
}\\end of main
```

Visiting all columns. Column followed by the next column. :]Comment [DS212

Re-set the value of small_odd for each new column :]Comment [DS213

Visiting all rows for the current column and check for the smallest odd number :]Comment [DS214

In each column, do 3 statements :]Comment [DS215

Perform sum for the smallest odd number in that column :]Comment [DS216

Syntax: Value-Returning function

The syntax of a value-returning function is:

```
functionType functionName(formal parameter list)
{
    statements
}
```

in which statements are usually declaration statements and/or executable statements. In this syntax, functionType is the type of the value that the function returns. The functionType is also called the data type or the return type of the value-returning function. Moreover, statements enclosed between curly braces form the body of the function.

Syntax: Formal Parameter List

The syntax of the formal parameter list is:

```
dataType identifier  dataType identifier,
```

Function Call

The syntax to call a value-returning function is:

```
functionName(actual parameter list)
```

Function	Header File	Purpose	Parameter(s) Type	Result
floor (x)	<cmath>	Returns the largest whole number that is not greater than x:floor(45.67) =45.00	double	double
islower(x)	<cctype>	Returns true if x is a lowercase letter; otherwise, it returns false;islower('h') is true	int	int
isupper(x)	<cctype>	Returns true if x is a uppercase letter; otherwise, it returns false; isupper ('K') is true	int	int
pow (x, y)	<cmath>	Returns xY; if x is negative, y must be a whole number: pow (0•16, 0•5) =0•4	double	double
sqrt(x)	<cmath>	Returns the nonnegative square root of x; x must be nonnegative: sqrt (4.0) 2.0	double	double
tolower(x)	<cctype>	Returns the lowercase value of x if x is uppercase; otherwise, it returns x	int	int
toupper(x)	<cctype>	Returns the uppercase value of x if x is lowercase; otherwise, it returns x	int	int

Function	Header File	Purpose	Parameter(s) Type	Result
abs (x)	<cmath>	Returns the absolute value of its argument: abs (-7) = 7	int (double)	int (double)
ceil (x)	<cmath>	Returns the smallest whole number that is not less than x: ceil(56.34) = 57.0	double	double
cos (x)	<cmath>	Returns the cosine of angle: x: cos(O.O) = 1.0	double (radians)	double
exp (x)	<cmath>	Returns ex, where e = 2•718 : exp(1.0) = 2.71828	double	double
fabs (x)	<cmath>	Returns the absolute value of its argument: fabs(-5.67) = 5.67	double	double

```
//How to use predefined functions.
#include <iostream>
#include <cmath>
#include <cctype>

using namespace std;

int main()
{
    int    x;
    double u, v;

    cout << "Line 1: Uppercase a is "
         << static cast<char>(toupper('a'))
         << endl; –                                    //Line 1

    u = 4.2;                                           //Line 2
    v = 3.0;                                           //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;           //Line 4

    cout << "Line 5: 5.0 to the power of 4 = "
         << pow(5.0, 4) << endl;                       //Line 5

    u = u + pow(3.0, 3);                               //Line 6
    cout << "Line 7: u = " << u << endl;               //Line 7

    x = -15;                                           //Line 8
    cout << "Line 9: Absolute value of " << x
         << "  " << abs(x) << endl;                    //Line 9

    return 0;
}
```

Sample Run:

```
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3    74.088
Line 5: 5.0 to the power of 4    625
Line 7: u = 31.2
Line 9: Absolute value of -15    15
```

This program works as follows. The statement in Line 1 outputs the uppercase letter that corresponds to 'a', which is A. Note that the function toupper returns an int value. Therefore, the value of the expression toupper('a' is 65, which is the ASCII value of 'A'.

To print A rather than 65, you need to apply the cast operator, as shown in the statement in Line 1. In the statement in Line 4, the function pow is used to outputuv. In C++ terminology, it is said that the function pow is called with the parameters u and v. In this case, the values of u and v are passed to the function pow. The other statements have similar meanings. N ote that the program includes the header files cctype and cmath, because it uses the functions toupper, pow, and abs from these header files.