2 Arrays

Significant Learning Outcomes

After the completion of this chapter, the learner

- identifies the scenarios where an array can be used.
- uses arrays to refer to a group of data.
- familiarises with the memory allocation for arrays.
- accesses any element in an array while solving problems.
- solves problems in which large amount of data is to be processed.
- represnts strings using character arrays.
- carries out various word processing operations using character arrays.

We use variables to store data in programs. But if the quantity of data is large, more variables are to be used. This will cause difficulty in accessing the required data. We have revised the concept of C++ data types in Chapter 1 and we used basic data types to declare variables and perform type conversion. In this chapter, a derived data type in C++, named 'array' is introduced. The word 'array' is not a data type name, rather it is a kind of data type derived from fundamental data types to handle large number of data easily. We will discuss the creation and initialisation of arrays, and operations like traversal.

2.1 Array and its need

An **array** is a collection of elements of the same type placed in contiguous memory locations. Arrays are used to store a set of values of the same type under a single variable name. Each element in an array can be accessed using its position in the list, called index number or subscript.

Why do we need arrays? We will illustrate this with the help of an example. Let us consider a situation where we need to store the scores of 20 students in a class and has to find their class

average. If we try to solve this problem by making use of variables, we will need 20 variables to store students' scores. Remembering and managing these 20 variables is not an easy task and the program may become complex and difficult to understand.

```
int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t;
float avg;
cin>>a>>b>>c>>d>>e>>f>>g>>h>>i>>j>>k>>l>>m>>n>>o>>p>>q>>r>>s>>t;
avg = (a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t)/20.0;
```

As it is, this code is fine. However, if we want to modify it to deal with the scores of a large number of students, say 1000, we have a very long and repetitive task at hand. We have to find a way to reduce the complexity of this task.

The concept of array comes as a boon in such situations. As it is a collection of elements, memory locations are to be allocated. We know that a declaration statement is needed for memory allocation. Let us see how arrays are declared and used.

2.1.1 Declaring arrays

Just like the ordinary variable, the array is to be declared properly before it is used. The syntax for declaring an array in C++ is as follows.

```
data_type array_name[size];
```

In the syntax, `data_type` is the type of data that the array variable can store, `array_name` is an identifier for naming the array and the `size` is a positive integer number that specifies the number of elements in the array. The following is an example:

```
int num[10];
```

The above statement declares an array named `num` that can store 10 integer numbers. Each item in an array is called an `element` of the array. The elements in the array are stored sequentially as shown in Figure 2.1. The first element is stored in the first location; the second element is stored in the second location and so on.

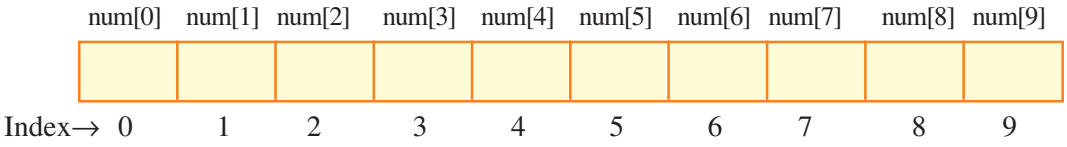

Fig. 2.1: Arrangement of elements in an array

Since the elements in the array are stored sequentially, any element can be accessed by giving the array's name and the element's position. This position is called the **index** or **subscript** value. In C++, the array index starts with zero. If an array is

declared as `int num[10];` then the possible index values are from 0 to 9. In this array, the first element can be referenced as `num[0]` and the last element as `num[9]`. The subscripted variable, `num[0]`, is read as "num of zero" or "num zero". It's a shortened way of saying "the num array subscripted by zero". So, the problem of referring the scores of 1000 students can be resolved by the following statement:

```
int score[1000];
```

The array, named `score`, can store the scores of 1000 students. The score of the first student is referenced by `score[0]` and that of the last by `score[999]`.

2.1.2 Memory allocation for arrays

The amount of storage required to hold an array is directly related to its type and size. Figure 2.2 shows the memory allocation for the first five elements of array **num**, assuming 1000 as the address of the first element. Since `num` is an integer type array, size of each element is 4 bytes (in a system with 32 bit integer representation using GCC) and it will be represented in memory as shown in Figure 2.2.

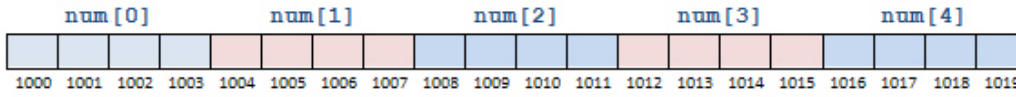

Fig. 2.2: Memory allocation for an integer array

The memory space allocated for an array can be computed using the following formula:

total_bytes = `sizeof`(array_type) × size_of_array

For example, total bytes allocated for the array declared as `float a[10];` will be 4 × 10 = 40 bytes.

2.1.3 Array initialisation

Array elements can be initialised in their declaration statements in the same manner as in the case of variables, except that the values must be included in braces, as shown in the following examples:

```
int score[5] = {98, 87, 92, 79, 85};
char code[6] = {'s', 'a', 'm', 'p', 'l', 'e'};
float wgpa[7] = {9.60, 6.43, 8.50, 8.65, 5.89, 7.56, 8.22};
```

Initial values are stored in the order they are written, with the first value used to initialize element 0, the second value used to initialize element 1, and so on. In the first example, `score[0]` is initialized to 98, `score[1]` is initialized to 87, `score[2]` is initialized to 92, `score[3]` is initialized to 79, and `score[4]` is initialized to 85.

If the number of initial values is less than the size of the array, they will be stored in the elements starting from the first position and the remaining positions will be initialized with zero, in the case of numeric data types. For `char` type array, such positions will be initialised with `' '` (space bar) character. When an array is initialized with values, the size can be omitted. For example, the following declaration statement will reserve memory for five elements:

```
int num[] = {16, 12, 10, 14, 11};
```

2. 1.4 Accessing elements of arrays

Array elements can be used anywhere in a program as we do in the case of normal variables. We have seen that array is accessed element-wise. That is, only one element can be accessed at a time. The element is specified by the array name with the subscript. The following are some examples of using the elements of the array named score:

```
score[0] = 95;
score[1] = score[0] - 11;
cin >> score[2];
score[3] = 79;
cout << score[2];
sum = score[0] + score[1] + score[2] + score[3] + score[4];
```

The subscript in brackets can be a variable, a constant or an expression that evaluates to an integer. In each case, the value of the expression must be within the valid subscript range of the array. An important advantage of using variable and integer expressions as subscripts is that, it allows sequencing through an array by using a loop. This makes statements more structured keeping away from the inappropriate usage as follows:

```
sum = score[0] + score[1] + score[2] + score[3] + score[4];
```

The subscript values in the above statement can be replaced by the control variable of `for` loop to access each element in the array sequentially. The following code segment illustrates this concept:

```
sum = 0;
for (i=0; i<5; i++)
     sum = sum + score[i];
```

An array element can be assigned a value interactively by using an input statement, as shown below:

```
for(int i=0; i<5; i++)
    cin>>score[i];
```

When this loop is executed, the first value read is stored in the array element `score[0]`, the second in `score[1]` and the last in `score[4]`.

Program 2.1 shows how to read 5 numbers and display them in the reverse order. The program includes two `for` loops. The first one allows the user to input array values. After five values have been entered, the second `for` loop is used to display the stored values from the last to the first.

Program 2.1: To input the scores of 5 students and display them in reverse order

```
#include <iostream>
using namespace std;
int main()
{
    int i, score[5];
    for(i=0; i<5; i++)   //  Reads the scores
    {
        cout<<"Enter a score: ";
        cin>>score[i];
    }
    for(i=4; i>=0; i--)   //  Prints the scores
        cout<<"score[" << i << "] is " << score[i]<<endl;
    return 0;
}
```

The following is a sample output of program 2.1:

```
Enter a score: 55
Enter a score: 80
Enter a score: 78
Enter a score: 75
Enter a score: 92
score[4] is 92
score[3] is 75
score[2] is 78
score[1] is 80
score[0] is 55
```

Accessing each element of an array at least once to perform any operation is known as ***traversal*** operation. Displaying all the elements of an array is an example of traversal. If any operation is performed on all the elements in an array, it is a case of traversal. Program 2.2 shows how traversal is performed in an array.

Program 2.2: Traversal of an array

```
#include <iostream>
using namespace std;
int main()
{
    int a[5], i;
    cout<<"Enter the elements of the array :";
    for(i=0; i<5; i++)
        cin >> a[i];  //Reading the elements
    for(i=0; i<5; i++)
        a[i] = a[i] + 1; // A case of traversal
    cout<<"\nNow value of elements in the array are...\n";
    for(i=0; i<5; i++)
        cout<< a[i]<< "\t";  // Another case of traversal
    return 0;
}
```

The following is a sample output of program 2.2:

```
Enter the elements of the array : 12  3   6   1   8
Now value of elements in the are...
13  4  7  2  9
```

Let us do

1. Write array declarations for the following:
   a. Scores of 100 students
   b. English letters
   c. A list of 10 years
   d. A list of 30 real numbers
2. Write array initialization statements for the following:
   a. An array of 10 scores: 89, 75, 82, 93, 78, 95, 81, 88, 77, and 82
   b. A list of five amounts: 10.62, 13.98, 18.45, 12.68, and 14.76
   c. A list of 100 interest rates, with the first six rates being 6.29, 6.95, 7.25, 7.35, 7.40 and 7.42.
   d. An array of 10 marks with value 0.
   e. An array with the letters of VIBGYOR.
   f. An array with number of days in each month.
3. Write a C++ code to input values into the array: `int ar[50];`
4. Write a C++ code fragment to display the elements in the even positions of the array: `float val[100];`

Let us solve another problem that requires traversal operation. Program 2.3 accepts five numbers from a user and finds the sum of these numbers.

Program 2.3: To find the sum of the elements of an array

```
#include <iostream>
using namespace std;
int main()
{
    int a[5], i, sum;
    cout<<"Enter the elements of the array :";
    for(i=0; i<5; i++)
        cin >> a[i]; //Reading the elements
    sum = 0;
    for(i=0; i<5; i++)
        sum = sum + a[i]; // A case of traversal
    cout<<"\nSum of the elements of the array is "<< sum;
    return 0;
}
```

The following is a sample output of Program 2.3:

```
Enter the elements of the array : 12  3   6   1   8
Sum of the elements of the array is 30
```

Program 2.4 illustrates another case of traversal to find the largest element in an array. In this program a variable `big` is used to hold the largest value. Initially it is assigned with the value in the first location. Then it is compared with the remaining elements. Whenever a larger value is found, it repalces the value of `big`.

Program 2.4: To find the largest element in an array

```
#include <iostream>
using namespace std;
int main()
{
    int a[5], i, big;
    cout<<"Enter the elements of the array :";
    for(i=0; i<5; i++)
        cin >> a[i];
    big = a[0];
    for(i=1; i<5; i++)
        if (a[i] > big)  // A case of traversal
            big = a[i];
```

```
    cout<<"\nThe biggest element is " << big;
    return 0;
}
```

The following is a sample output of program 2.4:

```
Enter the elements of the array : 12  3    6    1    8
The biggest element is 12
```

2.2 String handling using arrays

We know that string is a kind of literal in C++ language. It appears in programs as a sequence of characters within a pair of double quotes. Imagine that you are asked to write a program to store your name and display it. We have learned that variables are required to store data. Let us take the identifier `my_name` as the variable. Remember that in C++, a variable is to be declared before it is used. A declaration statement is required for this and it begins with a data type. Which data type should be used to declare a variable to hold string data? There is no basic data type to represent string data. We may think of **`char`** data type. But note that the variable declared using `char` can hold only one character. Here we have to input string which is a sequence of characters.

Let us consider a name "Niketh". It is a string consisting of six characters. So it cannot be stored in a variable of `char` type. But we know that an array of `char` type can hold more than one character. So, we declare an array as follows:

```
char my_name[10];
```

It is sure that ten contiguous locations, each with one byte size, will be allocated for the array named `my_name`. If we follow the usual array initialization method, we can store the characters in the string "Niketh" as follows:

```
char my_name[10]={'N', 'i', 'k', 'e', 't', 'h'};
```

Figure 2.3 shows the memory allocation for the above declared character array. Note that, we store the letters in the string separated by commas. If we want to input the same data, the following C++ statement can be used:

```
for (int i=0; i<6; i++)
    cin >> my_name[i];
```

During the execution of this statement, we have to input six letters of "Niketh" one after the other separated by **Space bar**, **Tab** or **Enter** key. The memory allocation in both of these cases will be as shown in Figure 2.3.

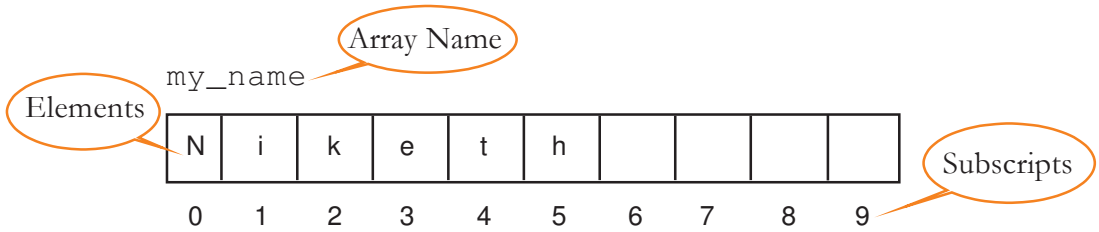

Fig. 2.3 : Memory allocation for the character array

So, let us conclude that a character array can be used to store a string, since it is a sequence of characters. However, it is true that we do not get the feel of inputting a string. Instead, we input the characters constituting the string one by one.

In C++, character arrays have some privileges over other arrays. Once we declare a character array, the array name can be considered as an ordinary variable that can hold string data. Let's say that a character array name is equivalent to a string variable. Thus your name can be stored in the variable `my_name` (the array name) using the following statement:

```
cin >> my_name;
```

It is important to note that this kind of usage is wrong in the case of arrays of other data types. Now let us complete the program. It will be like the one given in Program 2.5.

Program 2.5 To input a string and display

```
#include <iostream>
using namespace std;
int main()
{
     char my_name[10];
     cout << "Enter your name: ";
     cin >> my_name;
     cout << "Hello " << my_name;
     return 0;
}
```

On executing this program we will get the output as shown below.

```
Enter your name: Niketh
Hello Niketh
```

Note that the string constant is not `"Hello"`, but `"Hello "` (a white space is given after the letter `'o'`).

Let us do

Run Program 2.5 and input your full name by expanding the initials if any, and check whether the output is correct or not. If your name contains more than 10 characters, increase the size of the array as needed.

2.3 Memory allocation for strings

We have seen how memory is allocated for an array of characters. As Figure 2.3 shows, the memory required depends upon the number of characters stored. But if we input a string in a character array, the scene will be different. If we run Program 2.5 and input the string `Niketh`, the memory allocation will be as shown in Figure 2.4.

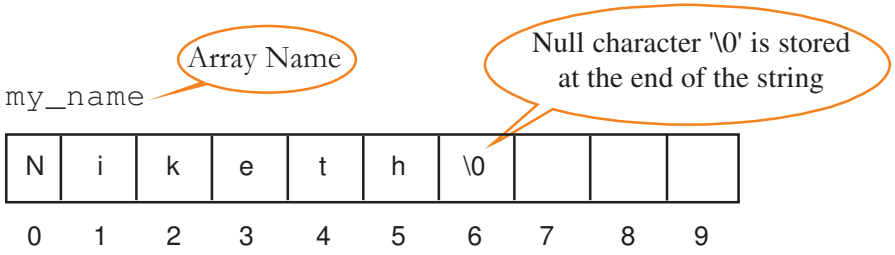

Fig. 2.4 : Memory allocation for the character array

Note that a null character `'\0'` is stored at the end of the string. This character is used as the string terminator and added at the end automatically. Thus we can say that memory required to store a string will be equal to the number of characters in the string plus one byte for null character. In the above case, the memory used to store the string `Niketh` is seven bytes, but the number of characters in the string is only six.

As in the case of variable initialization, we can initialize a character array with a string as follows:

```
char my_name[10] = "Niketh";
char str[] = "Hello World";
```

In the first statement 10 memory locations will be allocated and the string will be stored with null character as the delimiter. The last three bytes will be left unused. But, for the second statement, size of the array is not specified and hence only 12 bytes will be allocated (11 bytes for the string and 1 for `'\0'`).

2.4 Input/Output operations on strings

Program 2.5 contains input and output statements for string data. Let us modify the declaration statement by changing the size of the array to 20. If we run the program by entering the name `Maya Mohan`, the output will be as follows:

```
Enter your name: Maya Mohan
Hello Maya
```

Note that though there is enough size for the array, we get only the word "Maya" as the output. Why does this happen?

Let us have a close look at the input statement: `cin>>my_name;`. We have experienced that only one data item can be input using this statement. A white space is treated as a separator of data. Thus, the input `Maya Mohan` is treated as two data items, `Maya` and `Mohan` separated by white space. Since there is only one input operator (**>>**) followed by a variable, the first data (i.e., `Maya`) is stored. The white space after "Maya" is treated as the delimiter.

So, the problem is that we are unable to input strings containing white spaces. C++ language gives a solution to this problem by a function, named **`gets()`**. The function `gets()` is a console input function used to accept a string of characters including white spaces from the standard input device (keyboard) and store it in a character array.

The string variable (character array name) should be provided to this function as shown below:

```
gets(character_array_name);
```

When we use this function, we have to include the library file `cstdio.h` in the program. Let us modify Program 2.5, by including the statement `#include<cstdio>`, and replacing the statement `cin>>my_name;` by `gets(my_name);` After executing the modified program, the output will be as follows:

```
Enter your name: Maya Mohan
Hello Maya Mohan
```

The output shows the entire string that we input. See the difference between `gets()` and `cin`.

Though we do not use the concept of subscripted variable for the input and output of strings, any element in the array can be accessed by specifying its subscript along with the array name. We can access the first character of the string by `my_name[0]`, fifth character by `my_name[4]` and so on. We can even access the null character (`'\0'`) by its subscript. Program 2.6 illustrates this idea.

Program 2.6: To input a string and count the vowels in a string

```
#include <iostream>
#include <cstdio>    //To use gets() function
using namespace std;
int main()
{
    char str[20];
```

```
    int vow=0;
    cout<<"Enter a string: ";
    gets(str);
    for(int i=0; str[i]!='\0'; i++)
        switch(str[i])
        {   case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': vow++;
        }
    cout<<"No. of vowels in the string "<<str<<" is "<<vow;
    return 0;
}
```

If we run Program 2.6 by inputting the string "hello guys", the following output can be seen:

```
Enter a string: hello guys
No. of vowels in the string hello guys is 3
```

Now, let us analyse the program and see how it works to give this output.

- In the beginning, the `gets()` function is used and so we can input the string `"hello guys"`.
- The body of the `for` loop will be executed as long as the element in the array, referenced by the subscript `i`, is not the null character (`'\0'`). That is, the body of the loop will be executed till the null character is referenced.
- The body of the loop contains only a `switch` statement. Note that, no statements are given against the first four cases of the `switch`. In the last case, the variable `vow` is incremented by 1. You may think that this is required for all the cases. Yes, you are right. But, you should use the `break` statement for each case to exit the `switch` after a match. In this program the action for all the cases are the same and that is why we use this style of code.
- While the `for` loop iterates, the characters will be retrieved one by one for matching against the constants attached to the cases. Whenever a match is found, the variable `vow` is incremented by 1.
- As per the input string, matches occur when the value of `i` becomes 1, 4 and 7. Thus, the variable `vow` is incremented by 1 three times and we get the correct output.

We have seen how `gets()` function facilitates input of strings. Just like the other side of a coin, C++ gives a console function named **`puts()`** to output string data.

The function `puts()` is a console output function used to display a string data on the standard output device (monitor). Its syntax is:

```
puts(string_data);
```

The string constant or variable (character array name) to be displayed should be provided to this function. Observe the following C++ code fragment:

```
char str[10] = "friends";
puts("hello");
puts(str);
```

The output of the above code will be as follows:

```
hello
friends
```

Note that the string `"friends"` in the character array `str[10]` is displayed in a new line. Try this code using `cout<<"hello";` and `cout<<str;` instead of the `puts()` functions and see the difference. The output will be in the same line without a space in between them in the case of `cout` statement.

Let us do

Predict the output, if the input is "HELLO GUYS" in Program 2.6. Execute the program with this input and check whether you get the correct output. Find out the reason for difference in output. Modify the program to get the correct output for any given string.

Let us conclude

We have discussed array as a data type to refer to a group of same type of data. Memory allocation for arrays is explained with the help of schematic diagrams. The use of looping statements, especially `for` loop in manipulating the elements of an array are also illustrated through programs. We have also seen that how arrays help to handle strings effectively in programs.

Let us practice

1. Write a C++ program to input the amount of sales for 12 months into an array named SalesAmt. After all the input, find the total and average amount of sales.
2. Write a C++ program to create an array of N numbers, find the average and display those numbers greater than the average.

3. Write a C++ program to swap the first and the last elements of an integer array.
4. Write a C++ program to input 10 integer numbers into an array and determine the maximum and minimum values among them.
5. Write a C++ program to input a string and find the number of uppercase letters, lowercase letters, digits, special characters and white spaces.
6. Write a C++ program to count the number of words in a sentence.
7. Write a C++ program to find the length of a string.

Let us assess

1. The elements of an array with ten elements are numbered from ____ to ____.
2. An array element is accessed using _____.
3. If `AR` is an array, which element will be referenced using `AR[7]`?
4. Consider the array declaration `int a[3]={2,3,4};` What is the value of `a[1]`?
5. Consider the array declaration `int a[]={1,2,4};` What is the value of `a[1]`?
6. Printing all the elements of an array is an example for _____ operation.
7. Write down the output of the following code segment:

```
puts("hello");
puts("friends");
```

8. Write the initialisation statement to store the string "GCC".
9. Define an Array.
10. What does the declaration `int studlist[1000];` mean?
11. How is memory allocated for a single dimensional array?
12. Write C++ statements to accept an array of 10 elements and display the count of even and odd numbers in it.
13. Read the following statements:

```
char name[20];
cin>>name;
cout<<name;
```

What will be the output if you input the string "Sachin Tendulkar"? Justify your answer.

14. Write C++ statements to accept two single dimensional arrays of equal length and find the difference between corresponding elements.
15. Write a program to check whether a string is a palindrome or not.