

CHAPTER FIVE

ARRAYS AND STRINGS

Introduction

An **array** consists of a set of objects (called its **elements**), all of which are of the same type and are arranged contiguously in memory. In general, only the array itself has a symbolic name, not its elements. Each element is identified by an **index** which denotes the position of the element in the array. The number of elements in an array is called its **dimension**. The dimension of an array is fixed and predetermined; it cannot be changed during program execution.

Arrays are suitable for representing composite data which consist of many similar, individual items. Examples include: a list of names, a table of world cities and their current temperatures, or the monthly transactions for a bank account.

Defining Arrays

Like any other variable, array must be defined before it can be used to store data. The array definition comprises the *variable type* (which could be a **char**, an **int**, a **float**, etc;), the *name* of array and *size* in square brackets.

Syntax

Data_type ArrayName [size];

e.g int age[12];
 float grade[100];
 double angle[360];

The items in an array are called elements. The elements in an array should have same data type. Only their values vary.

E.g. int num[10]; (assuming this array has the elements 20, 30, 70, ..., 21, it looks the following in the computer's memory)

The first element in an array has an index value of 0, the second 1, and so on. If an array has 10 elements, then the first element has index value of 0 and the 10th element has an index value of 9.

2	num
0	[0]
3	num
0	[1]
7	num
0	[2]

6	num
4	[3]
2	num
9	[4]
3	num
7	[5]
1	num
6	[6]
1	num
9	[7]
2	num
6	[8]
2	num
1	[9]

Memory

Accessing Array Elements

To do some activities with the elements of the array, you have to access it first. Accessing the array elements consists of name of the array, square brackets, and the array index inside the square brackets.

e.g.

num[3] - access the fourth array element

num[0] - access the first array element.

num[9] - access the last array element.

Attempting to access a nonexistent array element (e.g., num[-1] or num[10]) leads to a serious runtime error (called 'index out of bounds' error).

Example 1

The following program stores 10 natural numbers in one array and displays them (with enough space between the numbers when displayed).

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main ( )
{
    int n[10];
    for(int i = 0;i<10;i++)
    {
        cout <<"enter natural number "<<i+1<<" ";
        cin>>n[i];
    }
    cout<<"Now displaying ";
```

```
    for(int j = 0;j<10;j++)
        cout<<setw(4)<<n[j];
    getch();
}
```

Initializing Arrays

It is possible to initialize the elements of the array during defining arrays. When initializing arrays there is no need to write the array size in the square brackets.

Syntax

```
Data_type ArrayName [ ] = {initializes};
```

```
e.g int nums[3] = {5, 10, 15};
```

initializes the three elements of nums to 5, 10, and 15, respectively. When the number of values in the initializer is less than the number of elements, the remaining elements are initialized to zero:

```
int nums[3] = {5, 10}; // nums[2] initializes to 0
```

When a complete initializer is used, the array dimension becomes redundant, because the number of elements is implicit in the initializer. The first definition of nums can therefore be equivalently written as:

```
int nums[] = {5, 10, 15}; // no dimension needed
```

The following program finds displays the sum of the initialized array elements.

Example 1

```
#include <iostream.h>
int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;
int main ()
{
    for ( n=0 ; n<5 ; n++ )
    {
        result += billy[n];
    }
    cout << result;
    return 0;
}
```

Example 2

The following program finds average of numbers that are stored in array.

```
#include<iostream.h>
```

```
#include<iomanip.h>
#include<conio.h>
void main ( )
{
    int num[10]= { 10,20,50,70,75,60,80,95,55,100};
    float sum = 0;
    float av;
    for(int i = 0;i<10; i++)
        sum+=num[i];
    av=sum/10;
    cout<<"The average is\t"<<av;
    getch();
}
```

Operations on Arrays

Each member of an array is a pseudo-variable and can be processed as such. This means that you can add the values of two members of the array(Number[2]+Number[0]), you can subtract the value of one of the members from another member(member[1]-Number[4]). In the same way, you can perform multiplication, division, or remainder operations on members of an array.

One of the regular operations performed on an array consists of adding the values of the members to produce a sum. Here is an example:

```
#include <iostream.h>
```

```
int main()
{
    // We know that we need a constant number of elements
    const int max = 10;
    int number[max];

    // We will calculate their sum
    int sum = 0;

    cout << "Please type 10 integers.\n";

    for( int i = 0; i < max; i++ )
    {
        cout << "Number " << i + 1 << ": ";
        cin >> number[i];
        sum += number[i];
    }

    cout << "\n\nThe sum of these numbers is " << Sum << "\n\n";
}
```

```
    return 0;  
}
```

This would produce:

Please type 10 integers.

Number 1: 120

Number 2: 42

Number 3: 75

Number 4: 38

Number 5: 904

Number 6: 6

Number 7: 26

Number 8: 55

Number 9: 92

Number 10: 20

The sum of these numbers is 1378

Another type of operation regularly performed on an array consists of looking for a value held by one of its members. For example, you can try to know if one of the members holds a particular value you are looking for. Here is an example:

```
#include <iostream.h>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    // Declare the members of the array
```

```
    int numbers[] = {8, 25, 36, 44, 52, 60, 75, 89};
```

```
    int find;
```

```
    int i, m = 8;
```

```
    cout << "Enter a number to search: ";
```

```
    cin >> find;
```

```
    for (i = 0; (i < m) && (Numbers[i] != Find); ++i)
```

```
        continue;
```

```
    // Find whether the number typed is a member of the array
```

```
    if (i == m)
```

```
        cout << find << " is not in the list" << endl;
```

```
    else
```

```
        cout << find << " is the " << i + 1
```

```
        << "th element in the list" << endl;
```

```
    return 0;
```

}

This would produce:

```
Enter a number to search: 44
44 is the 4th element in the list
```

One of the most regular operations performed consists of comparing the values of different members to get the lowest value of the members. Here is an example:

```
// Example of finding the minimum member of an array
#include <iostream.h>

int main()
{
    // The members of the array
    int numbers[] = {8, 25, 36, 44, 52, 60, 75, 89};
    int minimum = numbers[0];
    int a = 8;

    // Compare the members
    for (int i = 1; i < a; ++i) {
        if (numbers[i] < minimum)
            minimum = numbers[i];
    }
    // Announce the result
    cout << "The lowest member value of the array is "
         << minimum << "." << endl;
    return 0;
}
```

This would produce:

The lowest member value of the array is 8.

You can use this same approach to get the maximum value of the members of an array. Here is an example:

```
// Example of finding the maximum member of an array
#include <iostream.h>
int main()
{
    // The members of the array
    int numbers[] = {8, 25, 36, 44, 52, 60, 75, 89};
    int maximum = numbers[0];
    int a = 8;
```

```
// Compare the members
for (int i = 1; i < a; ++i)
{
    if (numbers[i] > maximum)
        maximum = numbers[i];
}
// Announce the result
cout << "The highest member value of the array is "
    << maximum << "." << endl;

return 0;
}
```

Arrays and Functions

An array can be passed to a function as argument. An array can also be returned by a function. To declare and define that a function takes an array as argument, declare the function as you would do for any regular function and, in its parentheses, specify that the argument is an array. You don't have to specify the dimension of the array. This means that you can leave the square brackets empty: Here is an example:

Example 1

```
#include <iostream.h>
void DisplayTheArray(double member[]);
int main()
{
    const int numberOfItems = 5;
    double distance [numberOfItems] = {44.14, 720.52, 96.08,
468.78, 6.28};

    cout << "Members of the array";
    DisplayTheArray(distance);

    return 0;
}

void DisplayTheArray(double member[])
{
    for(int i = 0; i < 5; ++i)
        cout << "\nDistance " << i + 1 << ": " << member[i];
}
```

```
    cout << endl;
}
```

This would produce: Members of the array

```
    Distance 1: 44.14
    Distance 2: 720.52
    Distance 3: 96.08
    Distance 4: 468.78
    Distance 5: 6.28
```

When you declare and define a function that takes an array as argument, if you plan to process the array, for example, if you want the calling function to control the number of elements to be processed, you should/must pass another argument that will allow the function to know how many members of the array would be considered. Such a function can be declared as follows:

Example2

```
#include <iostream.h>
void DisplayTheArray(double mbr[], int count);

int main()
{
    double distance[] = {44.14, 720.52, 96.08, 468.78, 6.28, 68.04,
364.55, 6234.12};

    // Processing 5 members of the array
    cout << "Members of the array";
    DisplayTheArray(distance, 5);

    // Processing all members of the array
    int sizeOfArray = sizeof(Distance)/sizeof(double);

    cout << "\nMembers of the array";
    DisplayTheArray(distance, sizeOfArray);

    return 0;
}

void DisplayTheArray(double member[], int counter)
{
    for(int i = 0; i < counter; ++i)
        cout << "\nDistance " << i + 1 << ": " << member[i];
    cout << endl;
}
```

This would produce:

Members of the array
 Distance 1: 44.14
 Distance 2: 720.52
 Distance 3: 96.08
 Distance 4: 468.78
 Distance 5: 6.28

Members of the array
 Distance 1: 44.14
 Distance 2: 720.52
 Distance 3: 96.08
 Distance 4: 468.78
 Distance 5: 6.28
 Distance 6: 68.04
 Distance 7: 364.55
 Distance 8: 6234.12

Multi Dimensional Arrays

An array may have more than one dimension (i.e., two, three, or higher). The organization of the array in memory is still the same (a contiguous sequence of elements), but the programmer's perceived organization of the elements is different. For example, suppose we wish to represent the average seasonal temperature for three major Ethiopia cities.

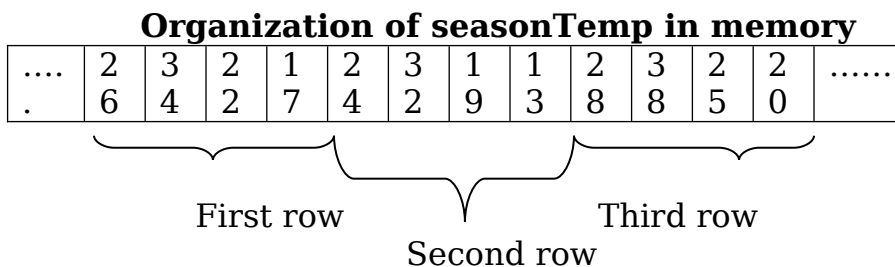
	Spring	Summer	Autumn	Winter
Addis Abeba	26	34	22	17
Adama	24	32	19	13
Awasa	28	38	25	20

This may be represented by a two-dimensional array of integers:

```
int seasonTemp[3][4];
```

The organization of this array in memory is as 12 consecutive integer elements. The

programmer, however, can imagine it as three rows of four integer entries each (see



Initializing Multidimensional Arrays

Like a one-dimensional arrays, it is also possible to initialize multidimensional arrays.

Syntax Datatype Name [value1][value2]---[valuen]
 ={ {x,y,z},
 { a,b,c},
 {d,e,f},
 :

`{l,j,k} };`

e.g.

```
int num [ ][3] = { {1,2,3},{4,5,6}};
```

As before, elements are accessed by indexing the array. A separate index is needed for each dimension. For example, Addis Abeba's average summer temperature (first row, second column) is given by `seasonTemp[0][1]`. The array may be initialized using a nested initializer:

```
int seasonTemp[3][4] = { {26, 34, 22, 17},  
                          {24, 32, 19, 13},  
                          {28, 38, 25, 20}};
```

Because this is mapped to a one-dimensional array of 12 elements in memory, it is equivalent to:

```
int seasonTemp[3][4] = {26, 34, 22, 17, 24, 32, 19, 13, 28, 38, 25, 20};
```

The nested initializer is preferred because as well as being more informative, it is more versatile. For example, it makes it possible to initialize only the first element of each row and have the rest default to zero:

```
int seasonTemp[3][4] = {{26}, {24}, {28}};
```

We can also omit the first dimension (but not subsequent dimensions) and let it be derived from the initializer:

```
int seasonTemp[][4] = {{26, 34, 22, 17},  
                       {24, 32, 19, 13},  
                       {28, 38, 25, 20}};
```

Example 1

The following program finds the highest temperature out of the three cities and the four seasons.

```
int main()
{
const int rows = 3;
const int columns = 4;
int seasonTemp[rows][columns] = { {26, 34, 22, 17},
                                   {24, 32, 19, 13},
                                   {28, 38, 25, 20}};

int highest = 0;
for (int i = 0; i < rows; ++i)
for (int j = 0; j < columns; ++j)
    if (temp[i][j] > highest)
    {
        highest = temp[i][j];
        cout<<"The highest temperature is =:"<<highest;
    }
}
```

Example 2

The following program stores sales of commodities in 5 stations for 3 months. You can also display what you stored for checkup purpose.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main ( )
{
    float sales[5][3];
    for(int s=0;s<5;s++)
        for (int m=0;m<3;m++)
            { cout <<"enter sales for station "<<s+1;
              cout<<" month "<<m+1<<" ";
              cin>>sales[s][m];
            }
    for (int j=0; j<=4; j++)
        for (int k=0;k<=2;k++)
            {cout<<" sales "<<j+1 <<" month "<<k+1;
              cout<<setw(4)<<sales[j][k];
              if (k= =2)
                  cout<<"\n";
            }
    getch( );
}
```

Strings

Since you have some understanding about arrays, it is very simple to examine strings. Strings are arrays of characters. In the arrays of characters, the null character is always added to the end of the characters. The null character is '\0'; with ASCII value of 0. In C++ there is no means to check the array bounds. Therefore, be careful when putting data into arrays not to exceed its maximum bounds.

For example the inbuilt C++ function *setw(bound)* helps to read characters as much as size of bound. When we display strings, characters inside an array are displayed until a null ('\0') character is encountered.

Examples

The following C++ program reads your name from keyboard and displays it.

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
void main ()
{ char name[20];
  cout<<"Enter your name \t";
  cin>>setw(20)>>name;
  cout<<name;
  getch();
}
```

If you need to read a string with blank spaces in between, the extraction operator>> considers the space to be a terminating character of a string. Thus, characters after a space will be ignored. To avoid this problem you can use the function *cin.get()*. This helps to read strings along with blank spaces.

e.g.

The following program reads your name and father's name and displays it.

```
#include<iostream.h>
#include<conio.h>
void main ( )
{
    char str [20];
    cout<<"Enter Full Name\t";
    cin.get(str,20);
    cout<<"your name and fathers name is \n"<< str;
}
```

Reading multiple lines of strings

Some time it is necessary to read multiple lines of string from the keyboard. The default terminating character is the new line ('\n') character. But it is possible to override the default character with some other characters so that we can read multiple lines of text until that terminating character is typed.

This can be achieved by adding the terminating character in the cin::get () function as an argument.

Example

The following program reads multiple lines of text and stores it in an array.

```
#include<iostream.h>
#include<conio.h>
void main ( )
{char str[100];
    cout<<"Enter a sstring and $ when finish \n";
    cin.get (str,100,'$');
    cout<<"you entered \n"<<str;
    getch ( ) ;
}
```

Arrays of strings

Like arrays of arrays, it is also possible to have arrays of strings. This is important to store arrays of names of people, or other strings for some other purpose. To define such constructions, we should write the size of the array and the maximum size of each string.

Example

The following program stores list of people in an array and display it.

```
#include<iostream.h>
#include<conio.h>

void main( )
{
    char name[5][10]={"kebede", "Ayele","Tufa","Almaz", "Kasa"};
    for (int i=0; i<5;i++)
```

```
    cout<<name[i]<<'\n';  
}
```

String Manipulation

strlen(): this c++ library function finds the length of a string, i.e. how many characters are in the string, excluding the null ('\0') character. This function takes the name of the string variable or constant as an input (argument) and returns an integer value. We must include the <string.h> header file to use this function.

Example

The following program takes your name and tells the length of your name.

```
#include<iostream.h>  
#include<string.h>  
#include<conio.h>  
void main()  
{  
    char name[25];  
    cout<<"Enter your name: ";  
    cin>>name;  
    cout<<"your name is "<<strlen(name)<<" characters long";  
    getch();  
}
```

strcpy(): It is a common process to copy one string into another. You can have many ways to copy a string. Copying a string character by character is one option to do so. It needs to access characters of a string character by character. Using the built in function *strcpy()* to copy a string is another easier method.

Example

```
#include<iostream.h>  
#include<string.h>  
#include<conio.h>  
void main ()  
{  
    const int SIZE=100;  
    char str1[SIZE];  
    char str2[SIZE];  
    cout<<"Enter a string\n";
```

```

    cin.get(str1,SIZE);
    strcpy(str2,str1);
    cout<<"Copy of your string is \n"<<str2;
    getch ();
}

```

strcpy() takes two arguments and copies the second variable (the right hand side) to the first (left hand side) variable.

strcat(): this function takes two arguments and concatenates (appends) the second variable to the first one.

Example

```

#include<iostream.h>
#include<conio.h>
#include<string.h>
void main ( )
{ const int p=30;
  char ch1[50],ch2[100]="I am proud ";
  cout<<ch2<<endl;
  cout<<"Enter a string to concatenate :";
  cin.get(ch1,50);
  strcat(ch2,ch1);
  cout<<"The concatenated is "<<ch2;
  getch();
}

```

strcmp(): this function takes two string arguments and compares them. The result of the comparison is an integer value of either negative, positive, or zero based on the following facts.

Example strcmp(str1,str2);

- 0 if str1 is the same as str2
- +ve (>0) if str2 comes first alphabetically
- -ve (<0) if str1 comes first alphabetically

Example: the following program takes two words from a user and compares their alphabetical precedence.

```

#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
  const int p=15;
  char ch1[p],ch2[p];
  cout<<"Enter first string: ";
  cin>>ch1;

```

```
cout<<"enter second string: ";
cin>>ch2;
if(strcmp(ch1,ch2) == 0)
    cout<<"The two words are the same";
else if(strcmp(ch1,ch2)>0)
    cout<<ch2<<" comes first alphabetically";
else if(strcmp(ch1,ch2)<0)
    cout<<ch1<<" comes first alphabetically";
    getch();
}
```

Examples

The following C++ program reads your name from keyboard and display it in reverse order.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<iomanip.h>
void main ( )
{ char name[20];
  cout<<"Enter your name \t";
  cin>>setw(20)>>name;
  for(int i=(strlen(name)-1);i>=0;i--)
      cout<<name[i];
  getch();
}
```

Exercise

1. Define two functions which, respectively, input values for the elements of an array of reals and output the array elements:

```
void ReadArray (double nums[], const int size);
void WriteArray (double nums[], const int size);
```

2. Define a function which reverses the order of the elements of an array of reals:

```
void Reverse (double nums[], const int size);
```

3. The following table specifies the major contents of four brands of breakfast cereals.

Define a two-dimensional array to capture this data:

	Fiber	Sugar	Fat	Salt	
Top Flake	12g	25g	16g	0.4g	
Cornabix	22g	4g	8g	0.3g	
Oatabix	28g	5g	9g	0.5g	

Ultrabran 32g 7g 2g 0.2g
Write a function which outputs this table element by element.

4. Write a C++ program that stores a 4 by 5 integer matrix and display it.
5. Write a C++ program that adds two matrices.