LECTURE-12

ARRAYS

12.1 What are arrays

So far we have been dealing with individual data elements of type int, float and char. If we have to store a vector which is nothing but a one dimensional matrix in computer memory than how do we proceed? Le tus take an example of a vector quantity. A vector is a collection of identical data objects stored under one name. Similarly an array is also a collection of identical data objects placed in contiguous memory locations that can be individually referenced by adding a name (index) to a unique identifier.

In this lecture we will deal with the method of declaring the array, array notation, array initialization and processing the arrays.

12.2 Array Notation and declaration

Individual values or data stored in an array are called elements. Array elements can be constants or variables.

Arrays are set of values of the same type, which have a single name followed by an index. In C++ a square bracket appears around the index right after the name.

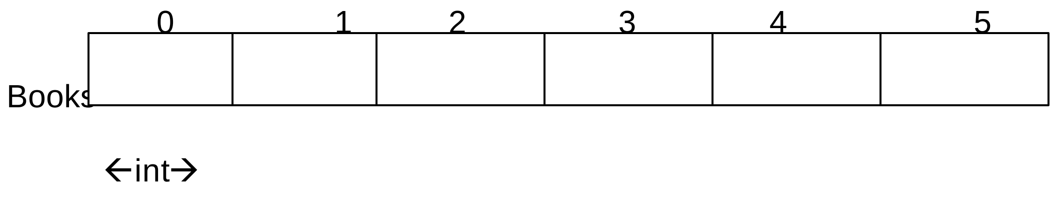
Declaring the name and type of an array and setting the number of elements in the array is known as *dimensioning* the array.

In general one dimensional array is expressed as:

storage_class data_type array_name[elements]

where *storage_class (*which is optional*)* refers to the scope of the array variable such as external, static, or an automatic; data_type refers to int, float, char etc.

For example, an array to contain 6 integer values of type int called books could be represented like this:



where each blank panel represents an element of the array, that in this case are integer values of type int. These elements are numbered from 0 to 5 since in arrays the first index is always 0, independently of its length.

Like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

type name [elements];

where type is a valid type (like int, float...), name is a valid identifier and the elements field (which is always enclosed in square brackets []), specifies how many of these elements the array has to contain.

Therefore, in order to declare an array called books as the one shown in the above diagram it is as simple as:

int books [6];

Let us write two programs to understand arrays. We ill first read and write ' n' number of data by simple program and then do the same operation using arrays.

Program 12.1 :Storing data without using array.

```
//reading data without arrays
#include <iostream>
#include <conio.h>
using namespace std;
void main()
{
int a=20,b=30,c=40,d=50;
cout<<a<<'\t'<<b<<'\t'<<c<<'\t'<<d;
getch();
}
//The output is 20 30 50
```

Program 12.2 :Storing data with array.

```
//reading data with arrays
#include <iostream>
#include <conio.h>
using namespace std;
void main()
{
int a[50],i,n;
cout<<"\n How many number u want to enter:";
cin>>n;
cout<<"\n Enter the "<<n<<"different elements in the array";
for (i =1; i<=n; i++)
{
cin>>a[i];
}
cout <<"\n Different array elements are:\n";
for (i =1; i<=n; i++)
{
```

```
cout<<'\t'<<a[i];
} getche();
}
/*
How many number u want to enter:5
Enter the 5different elements in the array10 20 30 40 50 60 70
Different array elements are:
10 20 30 40 50
*/
```

In the first program we have to write code for reading and writing each data. But in second program we assigned five spaces in the array and using for loop, entered seven data with space in between them. But since for loop reads only five data, these are stored in array a[5] in compact form. In second for loop a[5] is read in the same way. In the beginning initializing a[50] means we can store maximum 50 numbers in the array a[ ].

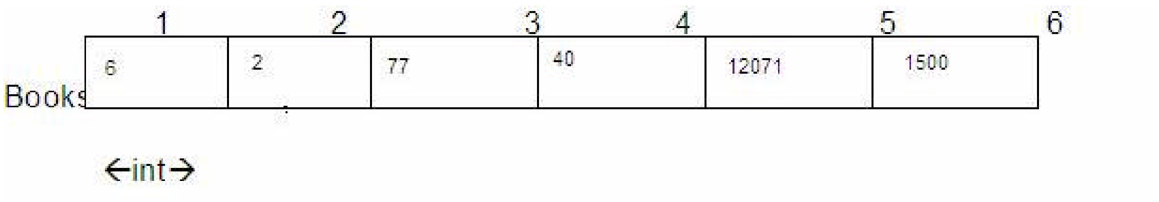
12.3 Initializing arrays.

When declaring a regular array of local scope (within a function, for example), if we do not specify otherwise, its elements will not be initialized to any value by default, so their content will be undetermined until we store some value in them. The elements of global and static arrays, on the other hand, are automatically initialized with their default values, which for all fundamental types, this means they are filled with zeros.

In both cases, local and global, when we declare an array, we have the possibility to assign initial values to each one of its elements by enclosing the values in braces { }. For example:

```
int books [6] = { 16, 2, 77, 40, 12071, 1500 };
```

This declaration would have created an array like this:



The amount of values between braces { } must not be larger than the number of elements that we declare for the array between square brackets [ ].

When an initialization of values is provided for an array, C++ allows the possibility of leaving the square brackets empty [ ]. In this case, the compiler will assume a size for the array that matches the number of values included between braces { }:

```
int books [ ] = { 16, 2, 77, 40, 12071, 1500 };
```

After this declaration, array books would be 6 ints long, since we have provided 6 initialization values.

12.4 Accessing the values of an array.

In any point of a program in which an array is visible, we can access the value of any of its elements individually as if it was a normal variable, thus being able to both read and modify its value. The format is as simple as:

```
name[index]
```

Following the previous examples in which books had 6 elements and each of those elements was of type int, the name which we can use to refer to each element is the following:

```
int books [6] = { 16, 2, 77, 40, 12071, 1500 };
```

The results of each of the above array element are:

books [0] = 16

books [1] = 2

books [2] = 77

books [3] = 40

books [4] = 122071

books [5] = 1500

For example, to store the value 75 in the third element of books, we could write the following statement:

```
books[2=] =  75;
```

and, for example, to pass the value of the third element of books to a variable called a, we could write:

a = books[2];

Therefore, the expression books[2] is for all purposes like a variable of type int.

Notice that the third element of books is specified books[2], since the first one is books[0], the second one is books[1], and therefore, the third one is books[2]. By this same reason, its last element is books[5]. Therefore, if we write books[6], we would be accessing the seventh element of `books` and therefore exceeding the size of the array.

At this point it is important to be able to clearly distinguish between the two uses that brackets `[ ]` which we have related to arrays. They perform two different tasks: one is to specify the size of arrays when they are declared; and the second one is to specify indices for concrete array elements. Do not confuse these two possible uses of brackets `[ ]` with arrays.

```
int books[6];         // declaration of a new array
books[2] = 75;    // access to an element of the array
```

If you read carefully, you will see that a type specifier always precedes a variable or array declaration, while it never precedes an access.

Some other valid operations with arrays:

```
 books[0] = a;
books[a] = 75;
b = books [a+2];
books[books[a]] = books[2] + 5;
```

Example 12.3

```
// arrays example
#include <iostream>
using namespace std;

int books [] = {16, 2, 77, 40, 12071};
int n, result=0;
int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += billy[n];
  }
  cout << result;
  return 0;
}
/*
12206 */
```

12.5 Multidimensional arrays

Multidimensional arrays can be described as "arrays of arrays". For example, a bi-dimensional array can be imagined as a bi-dimensional table made of elements, all of them of a same uniform data type. In another words, a two dimensional array is nothing but data expressed I matrix form as follows:

$$matrix = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

. All the members of above array named matrix are integers . Thus the above matrix can be declared in array in C++ would be:

int matrix [3][3];

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:

matrix [1][2]=a_{12}

(remember that array indices always begin by zero). Let us write a program to store a matrix in computer memory.

Program 12.4 : Program to store a matrix

```
//reading data of a two dimensional arrays
#include <iostream>
#include <conio.h>
using namespace std;
void main()
{
int a[10][10],n,m,i,j;
cout<<"\nEnter the number of rows and column of a matrix:";
cin>>n>>m;
cout<<"\nEnter the array elements:";
for(i=1; i<=n; i++)
(
for (j=0; j<=m; j++)
{
cin>>[i][j];
}
}cout<<"\nEntered matrix is:";
for(i=1; i<=n; i++)
(
for (j=0; j<=m; j++)
{
Cout<<"\t"<<[i][j];
}
cout<<"\n";
}
getche();
}
/*OUTPUT IS:
Enter the number of rows and column of a matrix : 2 3
Enter the array elements:2 3 4 2 5 1
Entered matrix is:
2   3   4
2   5   1
```

Multidimensional arrays are not limited to two indices (i.e., two dimensions). They can contain as many indices as needed. But be careful! The amount of memory needed for an array rapidly increases with each dimension. For example:

```
char century [100][365][24][60][60];
```

declares an array with a char element for each second in a century, that is more than 3 billion chars. So this declaration would consume more than 3 gigabytes of memory!

So far we discussed arrays which are of integer type. Arrays can be of float and char type too. The arrays with elements as float are written as

```
float  array_name [10 ];
```

and of char type can be

```
 char array_name [10];
```

Supposing we want to store a word AMRITSAR in an array how do e store? If we declare an array as

char city [9] . We can store only 8 character in the array city as in case of array of type char, one has to leave last cell as null cell, i.e. end of the string has null character, denoted by “\0”. Thus Amritsar is stored in the array city as:

city

A	0
M	1
R	2
I	3
T	4
S	5
A	6
R	7
‘\0’	8

12.6 Arrays as parameters

At some moment we may need to pass an array to a function as a parameter. In C++ it is not possible to pass a complete block of memory by value as a parameter to a function, but we are allowed to pass its address. In practice this has almost the same effect and it is a much faster and more efficient operation.

In order to accept arrays as parameters the only thing that we have to do when declaring the function is to specify in its parameters the element type of the array, an identifier and a pair of void brackets []. For example, the following function:

```
void procedure (int arg[])
```

accepts a parameter of type "array of int" called arg. In order to pass to this function an array declared as:

```
int myarray [40];
```

it would be enough to write a call like this:

```
procedure (myarray);
```

Here you have a complete example:

Program 12.5:

```
// arrays as parameters
#include <iostream>
using namespace std;

void printarray (int arg[], int length) {
  for (int n=0; n<length; n++)
    cout << arg[n] << " ";
  cout << "\n";
}

int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
  return 0;
}
```

```
5 10 15
2 4 6 8 10
```

As you can see, the first parameter (int arg[]) accepts any array whose elements are of type int, whatever its length. For that reason we have included a second parameter that tells the function the length of each array that we pass to it as its first parameter. This allows the for loop that prints out the array to know the range to iterate in the passed array without going out of range.

In a function declaration it is also possible to include multidimensional arrays. The format for a tridimensional array parameter is:

```
base_type[][depth][depth]
```

for example, a function with a multidimensional array as argument could be:

```
void procedure (int myarray[][3][4])
```

Notice that the first brackets [] are left blank while the following ones are not. This is so because the compiler must be able to determine within the function which is the depth of each additional dimension.

Program 12.6

```
// Program to find biggest number in an array
#include <iostream>
#include <conio.h>
using namespace std;
void main()
{
      int a[10],i,n,big;
      cout<< " Entr the number of elements:";
      cin>>n;
      cout<<"Enter the elements:";
      for (i=0; i<n; i++)
            cin>>a[i];
      big=a[0];
     for (i=0; i<n; i++)
       {if(a[i]>big)
       big=a[i];
       }
       cout<<"\nBiggest number is:"<<big;
       getch();
}
/* Entr the number of elements:5
Enter the elements:10 20 30 67 89

Biggest number is:89 */
```

Program 12.7:

```
//Program to sort an array using Bubble sort method
#include <iostream>
#include <conio.h>
using namespace std;

void main()
{
      int a[10],i,j,n,temp;
```

```
 cout<< " Enter the number of elements:";
 cin>>n;
 cout<<"Enter the elements:";
 for (i=0; i<n; i++)
       cin>>a[i];
   for (j=0; j<n; j++)
  {
        if(a[j]>a[j+1])
        {
        temp=a[j];
        a[j]=a[j+1];
        a[j+1]=temp;
        }
   }
}
   cout<<"\narray after sorting is:";
   for(i=0; i<n;i++)
         cout<<endl<<a[i];
   getch();
   }
```