Pemrograman Dasar

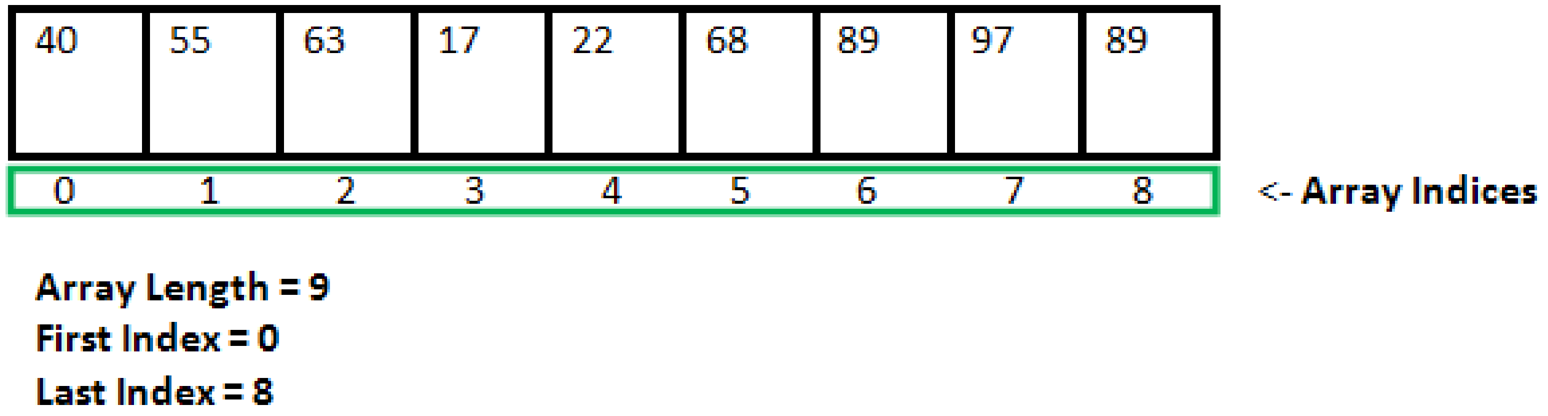
Week 11 : ***Array***

OUTLINES

1. Array Concept
2. Multi-dimensional arrays
3. Passing arrays to functions

1. Array Concept

C Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.



Why Do We Need Arrays?

We can use normal variables (v1, v2, v3,..) when we have small number of objects, but if we want to store large number of instances, it becomes difficult to manage them with normal variables. The idea of array is to represent many instances in one variable.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

```
type arrayName [ arraySize ];
```

This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement −

```
double balance[10];
int n = 10;
int arr2[n];
```

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ]. Therefore, if you write −

```
double balance[] = {10.0, 2.0, 3.4, 7.0, 50.0};
// compiler creates an array of size 5
// Above is same as double balance[5] = {10.0, 2.0, 3.4, 7.0, 50.0};

int arr[6] = {10, 20, 30, 40}
// Compiler creates an array of size 6, initializes first
// 4 elements as specified by user and rest two elements as 0.
// above is same as int arr[] = {10, 20, 30, 40, 0, 0};
```

Accessing Array Elements

Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1. Following are few examples.

```
int main() {
  int arr[5];
  arr[0] = 5;
  arr[2] = -10;
  arr[3/2] = 2; // this is same as arr[1] = 2
  arr[3] = arr[0];
  printf("%d %d %d %d", arr[0], arr[1], arr[2], arr[3]);
  return 0;
}
```

Output:

```
5 2 -10 5
```

Few Key Notes

1. Array size must be a constant value.
2. Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
3. It is a best practice to initialize an array to zero or null while declaring, if we don’t assign any values to array. (ProgDas11-1-1.c)
4. Arrays have 0 as the first index not 1.
5. If the size of an array is n, to access the last element, (n-1) index is used.
6. Suppose the starting address of mark[0] is 2120d. Then, the next address, a mark[1], will be 2124d, address of mark[2] will be 2128d and so on. It's because the size of a float is 4 bytes. (ProgDas11-1-2.c)

Few Key Notes

7. There is no index out of bound checking in C. The program won’t compile in C++. (ProgDas11-1-3.c)
8. In C, it is not compiler error to initialize an array with more elements than specified size. If we save the above program as a .cpp, the program generates compiler error “error: too many initializers for ‘int [2]'” (ProgDas11-1-4.c)

Example

ProgDas11-1-5.c

Exercise 1 **(ProgDas11-1-6.c)**

Create a program to find an average number that you inserted manually

Output:

```
Enter n: 5
Enter number1: 45
Enter number2: 35
Enter number3: 38
Enter number4: 31
Enter number5: 49
Average = 39
```

Exercise 2 (ProgDas11-1-7.c)

Program to Find Largest Element of an Array

Output:

```
Enter total number of elements(1 to 100): 3
Enter Number 1: 23.4
Enter Number 2: -34.5
Enter Number 3: 50
Largest element = 50
```

2. Multi-dimensional arrays

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional integer array −

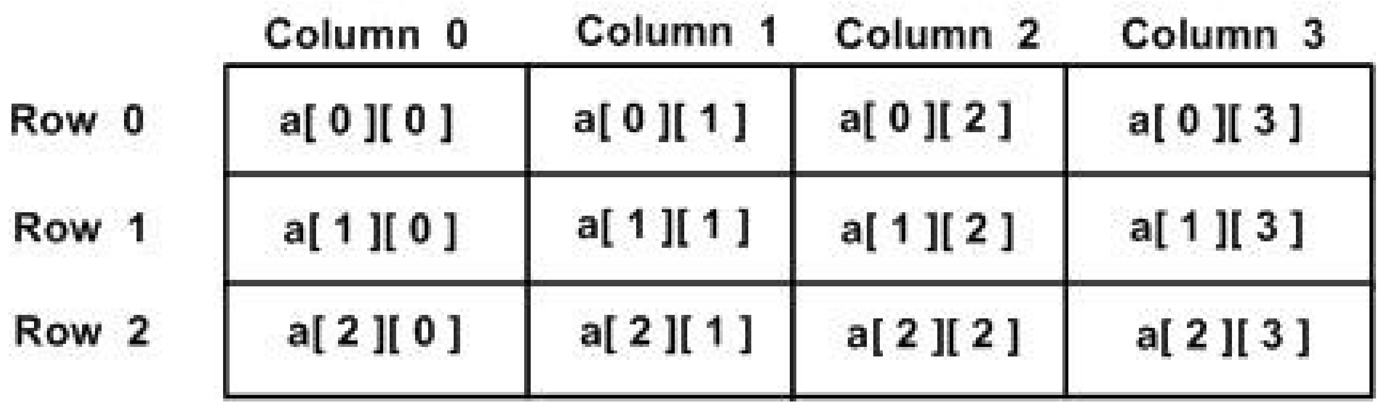
```
int threedim[5][10][4];
```

Two-dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows −

```
type arrayName [ x ][ y ];
```

Where type can be any valid C data type and arrayName will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array a, which contains three rows and four columns can be shown as follows −



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Every element in the array a is identified by an element name of the form a[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example −

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array.

In C/C++, initialization of a multidimensional arrays must be specified.

For example, following program fails in compilation because two dimensions are not specified

```
int main()
{
  int a[][][2] = { {{1, 2}, {3, 4}},
                   {{5, 6}, {7, 8}}
                 };  // error
  printf("%d", sizeof(a));
  getchar();
  return 0;
}
```

Following the programs work without any error.

```
#include<stdio.h>
int main()
{
  int a[][2] = {{1,2},{3,4}}; // Works
  printf("%d", sizeof(a)); // prints 4*sizeof(int)
  getchar();
  return 0;
}
```

Example

ProgDas11-2.c

Exercise 3 (ProgDas11-2-1.c)

Program to Sum of two matrices using Two dimensional arrays

Output

```
Enter elements of 1st matrix
Enter a11: 2;
Enter a12: 0.5;
Enter a21: -1.1;
Enter a22: 2;
Enter elements of 2nd matrix
Enter b11: 0.2;
Enter b12: 0;
Enter b21: 0.23;
Enter b22: 23;
Sum Of Matrix:
2.2     0.5
-0.9    25.0
```

Exercise 4 (ProgDas11-2-2.c)

Program to Find Transpose of a Matrix

Output

```
Enter rows and columns of
matrix: 2

3

Enter element of matrix:

Enter element a11: 2

Enter element a12: 3

Enter element a13: 4

Enter element a21: 5

Enter element a22: 6

Enter element a23: 4


Entered Matrix:

2   3   4

5   6   4


Transpose of Matrix:

2   5

3   6

4   4
```

3. Passing arrays to functions

Passing One-dimensional Array to a Function

Passing a single element of an array to a function is similar to passing variable to a function.

```
#include <stdio.h>
void display(int age)
{
    printf("%d", age);
}

int main()
{
    int ageArray[] = {2, 3, 4};
    display(ageArray[2]); //Passing array element ageArray[2]
    return 0;
}
```

Passing an Entire Array to a Function

If you want to pass a single-dimension array as an argument in a function, you would have to declare a formal parameter in one of following **three ways** and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received.

To pass an entire array to a function, only the name of the array is passed as an argument. However, notice the use of [] after argument name in float average(float age[]). This informs the compiler that you are passing a one-dimensional array to the function.

Way-1

Formal parameters as a pointer –

```
void myFunction(int *param) {
    .
    .
    .
}
```

Way-2

Formal parameters as a sized array –

```
void myFunction(int param[10]) {
   .
   .
   .
}
```

Way-3

Formal parameters as an unsized array –

```
void myFunction(int param[]) {
   .
   .
   .
}
```

Exercise 4 **(ProgDas11-3.c)**

From the program or source code 11-1-6.c, separate into two function that consist of `main()` and `getAverage()`.

Pass the array data that has been initialized from function `main()` and calculate average of the array data in function `getAverage()`, and then return the value that will be obtained in function `main()`.

Passing Multi-dimensional Arrays to Function

To pass multidimensional arrays to a function, only the name of the array is passed (similar to one dimensional array).

Example

ProgDas11-3-1.c

4. Three Dimensional Array

Example

ProgDas11-6.c

THANKYOU