**Array Declaration**

The syntax of array declaration is as follows,

{<storage class>} data type <array\_name>[expression]{[expression]};

In the above, curly braces indicate optional parts of the declaration.

Example:int arr[10];

The above declaration defines an array called *arr* of size 10, i.e., *arr* consists of 10 elements of same data type *int*. The elements occupy consecutive cells in the memory (each cell houses one element) and forms an ordered set of elements. Each element can be identified by its position in the array, and is also referred as the subscript of the array. The first element is at position 0 and nth element can be found in the (n -1)th position.

The name of the array is *arr*, which contains the address of the first element (i.e. &arr[0]) of the array. However, an array name such as *arr* differs from an ordinary pointer variable (likeint \*p;), because, it is static in nature and cannot point to a new memory location other than what it is pointing to by virtue of the declaration.

http://www.how2lab.com/be/uploads/imglib/c/array1.gif

More examples of array declaration:

static float grade[10];

char name[20];

Array Manipulation

The most convenient way of performing array manipulation is to use the for repetitive construct (for loop) for accessing elements of the array.

The following example illustrates the usage of an array in implementing addition of two vectors:

#include <stdio.h>

#define dimension 100

typedef int vector[dimension];

main()

{

vector vect\_1, vect\_2, result\_vect;

int i,n;

printf("Enter the Vector dimension: ");

scanf("%d", &n); fflush(stdin);

/\* Accept values for the two arrays vect\_1 & vect\_2 \*/

for(i=0; i < n; i++)

{

printf("Give Vector element #%d: ", i+1);

scanf("%d %d", &vect\_1[i], &vect\_2[i]); fflush(stdin);

result\_vect[i] = 0; //initialize elements of array result\_vect to 0

}

/\* Add vectors and create resultant vector \*/

for(i=0; i < n; i++)

result\_vect[i] += (vect\_1[i] + vect\_2[i]);

/\* Display the resultant matrix \*/

printf("\n\n");

for(i = 0; i < n; i++)

{

printf("%d\t", result\_vect[i]);

}

printf("\n");

}

How to initialize array elements during declaration?

Array initialization can be performed in the following way:

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

Note, however, that this definition-cum-initialization is permitted only in case of external variable definition. To include such initialization statement inside a function, storage class clause static has to be used as a prefix. See examples below.

Describe the output generated by each of the following programs.

#include <stdio.h>

main()

{

int i;

int a, sum = 0;

//declaring & initializing within a function

static int x[10] = {9,8,7,6,5,4,3,2,0};

for(i = 0; i < 10; ++i)

if((i % 2) == 0)

sum += x[i];

printf("%d", sum);

}

#include <stdio.h>

#define ROWS 3

#define COLS 4

//external declaration & initialization

int x[ROWS][COLS] = {12,11,10,9,8,7,6,5,4,3,2,1};

main()

{

int i, j, max;

for(i = 0; i < ROWS; ++i)

{

max = 9999;

for(j = 0; j < COLS; ++j)

if(x[i][j] < max)

max = z[i][j];

printf("%d", max);

}

}

**Exercises**

Identify the errors in the following C program (if any) which initializes an array such that each of its ten elements is assigned with 0 value.

main()

{

int num\_arr[10], i = 0;

...

...

for(; ++i < 10; num\_arr[i] = 0);

}

Identify the array defined in each of the following statements. Indicate what values are assigned to the individual array elements.

char game[7] = {'C', 'R', 'I', 'C', 'K', 'E', 'T'};

char match[] = "Football";

Write an appropriate array definition for each of the following cases:

a. Define a one dimensional, integer array called A with 10 elements and initialize the array elements with 2, 5, 8, 11, ... , 29.

b. Create a one dimensional, four element character array called object and assign the characters 'C', 'I', 'R', 'C', 'L' and 'E' to the array elements.

c. Define a one dimensional, six element floating point array called flt\_const having following initials values - 2.005, -3.05452, -1e-4, 340.179, 0.3e8, 0.023415

**Lab work**

Write a C program that will accept a line of text as input, store it in an array, and then print it backwards. Assume that the length of the string cannot exceed 80 characters, and while entering the data, the string will be terminated by a carriage return. Test the program by entering a suitable message.

**Representing a string as a character array**

A string constant, enclosed within a pair of double quotes, consists of 0 (empty string) or more characters terminated by a null ('\0') character, which indicates the end of the string.

The following statement in C defines a string variable:

char name[21];

The above declaration allocates a 21 character space in memory which can store 20 characters each of one byte (with name[0] as the starting element). One extra byte (the last element) is used to store the null character. The null character, represented by \0 serves the purpose of indicating the end of string. The string name can be initialized during declaration, by a string constant.



Example:

#include <stdio.h>

main()

{

static char name[22] = "Ratindra Nath Bhaskar";

printf("%s\n", name);

printf("Enter a new name: ");

scanf("%s", name); fflush(stdin);

printf({"\nThe new name is %s\n", name);

}

The above program displays the content of the variable name i.e. Ratindra Nath Bhaskar and allows the user to accept a new name in the same variable, and finally comes out of the program displaying the new name on the screen. Notice the absence of the & operator as a prefix of the variable name in the scanf statement, which requires the address of the variable. This is because the variable name itself stores the address of the string variable. This will be more clear from the following figure which shows the representation of the array name in the memory.

The variable name is a static pointer which stores the address of the first element of the name array i.e &name[0]. A static pointer means it cannot be reassigned with a new value.

From the above discussion it is obvious that a string variable requires only the starting address of the memory location where the string constant is located, because the end of the string is always indicated by the null character (\0).

To appreciate how this fact is utilized, look at the following example, where characters are read one by one from the console and stored in a characer array.

#include <stdio.h>

#define size 100

char name[size];

main()

{

int count = 0;

char c;

while((c = getchar()) != '\n')

{

name[count] = c;

++count;

}

name[ count] = '\0';

}

**Passing array to a function**

An array name can be used as an argument to a function, thus permitting the entire array to be passed to the invoked function. The array name must appear by itself (without brackets or subscript) as an actual argument during the function call. The function's formal parameter is also written in the same manner, though it must be declared as an array within the formal parameter declaration. When a one dimensional array is used as a formal parameter, in the function declaration array name is written with a pair of empty square brackets. The size of an array is not specified within the formal parameter declaration. Since array name identifies the address of the first element, specifying array name as a parameter to a function essentially implements call by reference mode of parameter transfer.

Example:

A C program to read a set of numbers and sort them. It sorts a one dimensional integer array in ascending order.

#include <stdio.h>

#define dimension 100

main()

{

int i, k, data[dimension];

void bubble\_sort(int m,int a[]);

void read\_data(int n, int x[]);

printf("\nEnter the number of elements to be sorted: ");

scanf("%d", &k); fflush(stdin);

read\_data(k, data);

bubble\_sort(k, data);

printf("\n\nSorted data elements are:\n");

for(i=0; i < k; i++)

printf("i = %d, data = %d\n", i+1, data[i]);

}

/\* Read an array of integer \*/

void read\_data(int n, int x[])

{

int j;

for(j=0; j < n ; j++)

{

printf("\nEnter data element #%d: ", j+1);

scanf("%d", &x[j]); fflush(stdin);

}

return;

}

/\* Sort an integer array a of m elements in ascending order \*/

void bubble\_sort(int m, int a[])

{

int i, j, temp;

int not\_sorted = 1; //not\_sorted is true when the array a is not sorted

j = m;

while(not\_sorted)

{

not\_sorted = 0; //assume sorted

for(i=0; i < j-1; ++i)

{

if(a[i] > a[i+1])

{

temp = a[i];

a[i] = a[i+1];

a[i+1] = temp;

not\_sorted = 1; //data was not sorted

}

}

--j;

} //repeat until sorted

return;

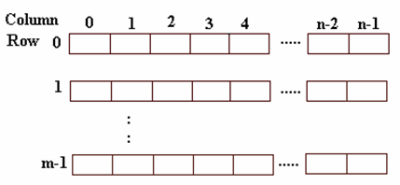
}

**Multidimensional arrays**

We have seen in the [earlier article](http://www.how2lab.com/programming/c/array1.php) that a one-dimensional array of size **n** is represented as a contiguous block of **n**number of elements. Likewise, a two dimensional array of size **m x n** (here **m** and **n** are representing the two dimensions) can be represented as **m** numbers of one-dimensional arrays each consisting of **n** number of elements.

The syntax for declaring a multidimensional array is as follows:

storage-class datatype array\_name [expression1] [expression2] ... [expression n];



Each element in the array is accessed by the combination of array name and a pair of subscripts indicating relative position of the element in the memory. The number of subscripts depend on the dimensions of the array.

Example 2-dimensional array declaration:

int x[3][3];

Example 2-dimensional array declaration along with initialization:

int data\_values[3][4]={

{1,2,3,4},

{5,6,7,8},

{10,11,12,13}

};

The following example of matrix manipulation illustrates the usage of multidimensional array.

#include <stdio.h>

#define row\_size 100

#define col\_size 100

typedef int matrix[row\_size][col\_size];

main()

{

matrix a,b,c;

int i,j,k,n;

printf("Enter the matrix dimension: ");

scanf("%d ",&n); fflush(stdin);

/\* accept values of elements of the two arrays a & b to form nxn matrices \*/

for(i=0; i < n; i++)

for(j = 0 ; j < n ; j++)

{

printf("Give matrix element with row %d column %d : ", i+1, j+1);

scanf("%d %d", &a[i][j], &b[i][j]); fflush(stdin);

c[i][j] = 0; //initialize elements of the 3rd array c to 0

}

/\* matrix multiplication and creation of the resultant matrix \*/

for(i = 0; i < n; i++)

for(j = 0; j < n; j++)

for(k = 0; k < n; k++)

c[i][j] += (a[i][k] \* b[k][j]);

/\* display the resultant matrix \*/

printf("\n\n");

for(i=0; i < n ; i++)

{

for(j = 0; j < n ; j++)

{

printf("%d\t",c[i][j]);

}

printf("\n");

}

}

**Exercise**

Identify the array defined in each of the following statements. Indicate what values are assigned to the individual array elements.

int a[2][4] = {

{5,4,3,2},

{9,8,7,6}

};

int data[2][3][4] = {

{

{3,2,1},

{9,8},

{5,4,6,7}

},

{

{1,2},

{},

{2,3,4}

}

};

char room\_colours[3][7] = {

{'B', 'l', 'u', 'e','\0'},

{'G', 'r', 'e', 'e', 'n','\0'},

{'Y', 'e', 'l', 'l', 'o', 'w','\0'}

};

**Passing multidimensional array as parameter to a function**

The formal argument declaration within a function definition must include explicit size specifications of all the subscripts except the first. The size specification must be consistent with the corresponding size specification in the calling program. The first subscript position should be written with an empty pair of square brackets as in one-dimensional array. See example below for more clarity.

#include <stdio.h>

#define row\_size 50

#define col\_size 50

main()

{

int a[row\_size][col\_size], b[row\_size][col\_size], c[row\_size][col\_size];

int m,n,k;

void read\_data(int x[][col\_size], int nrows, int ncols);

void matrix\_mult(int x[][col\_size], int y[][col\_size], int z[][col\_size], int nrows, int ncols1, int ncols2);

read\_data(a, m, n);

read\_data(b, n, k);

...

matrix\_mult(a, b, c, m, n, k);

...

}

void matrix\_mult(int x[][col\_size], int y[][col\_size], int z[][col\_size], int nrows,int ncols1, int ncols2)

{

int i,j,k;

for(i=0; i < nrows; ++i)

for(k=0; k < ncols2; ++k)

{

z[i][k] = 0;

for(j=0; j < ncols1; ++j)

z[i][k] += x[i][j] \* y[j][k];

}

return;

}

**Two dimensional character array**

A two-dimensional character array consists of a collection of single dimensional character array. Each 1-dimensional character array is a string. The declaration of a two-dimensional character array does not differ much from two-dimension array declaration of integer data, except that each individual string is terminated by a null character.

The syntax for declaring a two dimensional character array is as follows:

char name [3][20];

The variable name is a two dimensional string capable of storing 3 strings, where maximum length of each string is 19 characters (one position is kept aside for null). The initialization of such an array of strings can be performed in either of the two ways as shown below:

char city[4][10]={

{"Calcutta"},

{"Bombay"},

{"Madras"},

{"Delhi"}

};

The above way of initializing automatically adds a null character (\0) to the end of each string.

char city[4][10]={

{'C', 'a', 'l', 'c', 'u', 't', 't', 'a', '\0'},

{'B', 'o', 'm', 'b', 'a', 'y', '\0'},

{'M', 'a', 'd', 'r', 'a', 's', '\0'},

{'D', 'e', 'l', 'h', 'i', '\0'}

};

The above character array will be represented in the memory as shown below:

For beter understanding of how to use a 2-dimensional array of strings in a C program, observe the following sample program. Also try it out on your computer.

#include <stdio.h>

#define r 10

#define c 30

main()

{

char name[r][c], chr;

int i,j;

for(i=0; i < r; i++)

{

printf("Enter Name #%d: ", i+1);

j = 0;

while(((chr=getchar()) != ‘\n') && j< (c – 1))

{

name[i][j] = chr;

++j;

}

name[j] = '\0';

}

for(i=0; i < r; i++)

printf("Name %d is : %s\n", i+1, name[i]);

}

Exercise

Match the following :

i. arr\_ele[3] a. index

ii. subscript b. three dimensional array

iii. int num[3][4] c. fourth element of an array

iv. int num[4][3] d. two dimensional array with 3 rows and 4 columns

v. int num[3][4][5] e. two dimensional array with 4 rows and 3 columns

**Lab Work**

1.Write an interactive C program to process the marks obtained in a test for a group of students in the programming language course. The total number of tests to be conducted for the students is specified by the user at the beginning of the program. For each student in the class the program expects student's name and marks obtained in each test. It then calculates the average score for each student, and an overall class average (an average of the individuals student averages). The information to be displayed includes subject name, number of tests conducted and the overall class average, followed by the performance of each student that in turn includes the name of the student; the individual test scores and the average score for each student.

Hint: The student names should be stored in two-dimensional character array. A separate two dimensional floating point array should be used for storing the examination scores. The performances of the students are to be displayed as shown below:

Test scores (percent)

Name Test 1 Test 2 Test 3 Test 4 Test 5 Average

----------------------------------------------------------------------------------------------------

Mehul 45 80 80 95 55 75

Ravi 60 50 70 75 55 80

Raj 40 30 10 45 60 55

Pravin 0 5 5 0 10 5

----------------------------------------------------------------------------------------------------

Class Average 53.75

2.Consider the following list of States and their Capitals:

West Bengal Kolkata

Jharkhand Ranchi

Orissa Bhubaneswar

Maharashtra Mumbai

Kerala Thiruvanantapuram

Gujarat Gandhinager

Haryana Chandigarh

Uttar Paradesh Lucknow

Madhya Paradesh Bhopal

Himachal Paradesh Shimla

Write an interactive C program that will accept the name of a state as input and print the corresponding capital, and vice-versa. Entering the word End as input will terminate the program.

There are 500 light bulbs (numbered 1 to 500) arranged in a row. Initially, they are all OFF (indicating by 0). Starting with bulb 2, all even numbered bulbs are turned ON. Next starting with bulb 3, and visiting every third bulb, it is turned ON if it is OFF, and it is turned OFF if it is ON. This procedure is repeated for every fourth bulb, every fifth bulb, and so on up to the 500th bulb. Write a program to determine which bulbs are OFF at the end of above procedure.

Write a C function called merge() to merge two sorted arrays so that the merged data is also sorted. Use merge function recursively to sort an array of unsorted data.