iMSCIT

SEMESTER:I

1601102

PROBLEM SOLVING THROUGH PROGRAMMING

UNIT: V

Unit 5: Introduction to Arrays and Storage Classes

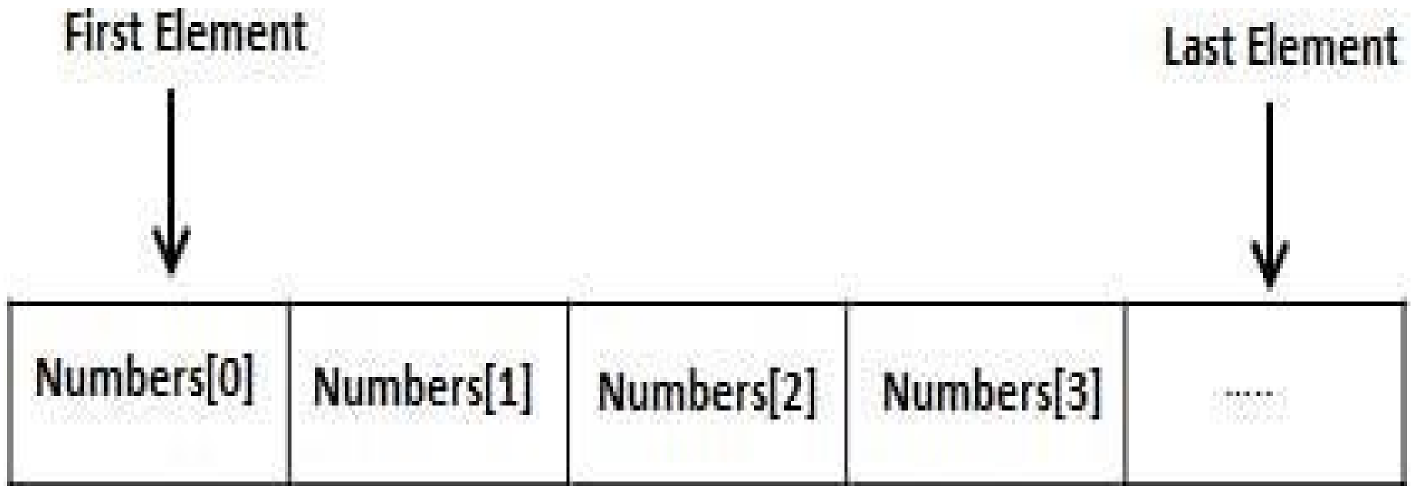
- **Introduction to Arrays**
- **Array Definition**
- **Array Dimension**
- **Types of Array**
  - Single-Dimensional Array
  - Multidimensional Array
  - Three Dimensional Array
- **Operations on Single-Dimensional Array**

Introduction to Array

- **An array is defined as the collection of similar type of data items stored at contiguous memory locations.**
- Arrays a **kind of data structure that can store a fixed-size sequential collection of elements of the same type.**
- **Arrays are the derived data type** in C programming language which can store the primitive type of data such as int, char, double, float, etc.
- The array is the simplest data structure where **each data element can be randomly accessed by using its index number.**
- 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

Introduction to Array

- All arrays consist of contiguous memory locations. **The lowest address corresponds to the first element and the highest address to the last element**.
- **Starting index is knows as Base Address or Base index.**
- **Array always starts from index 0.**



Properties of Array

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantages and Disadvantage of Array

Advantages:

1) Code Optimization: Less code to the access the data.

2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.

3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.

4) Random Access: We can access any element randomly using the array.

Disadvantages:

Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically

Array Terminology

Size

- Number of elements or capacity to store elements in an array.

Type

- Refer to data type.

Base

- The address of first element (0th) is a base address.

Index

- The array name is used to refer to the array element, num[x]

Range

- Index of an array

Declaration of C Array

Syntax:

```
data_type array_name[array_size];
```

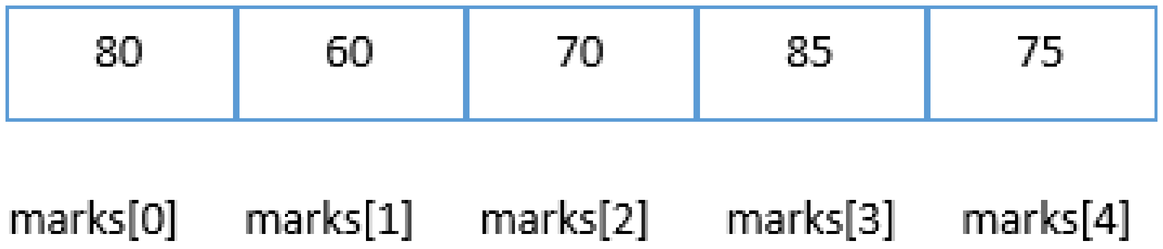
Example:

```
int marks[5];
```

Initialization of C Array

- **We can also initialize each element of the array by using the index. Consider the following example.**

```
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



Initialization of Array

Access element of an array

- **An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −**

```
int a=marks[1];
```

C Array: Declaration with Initialization

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

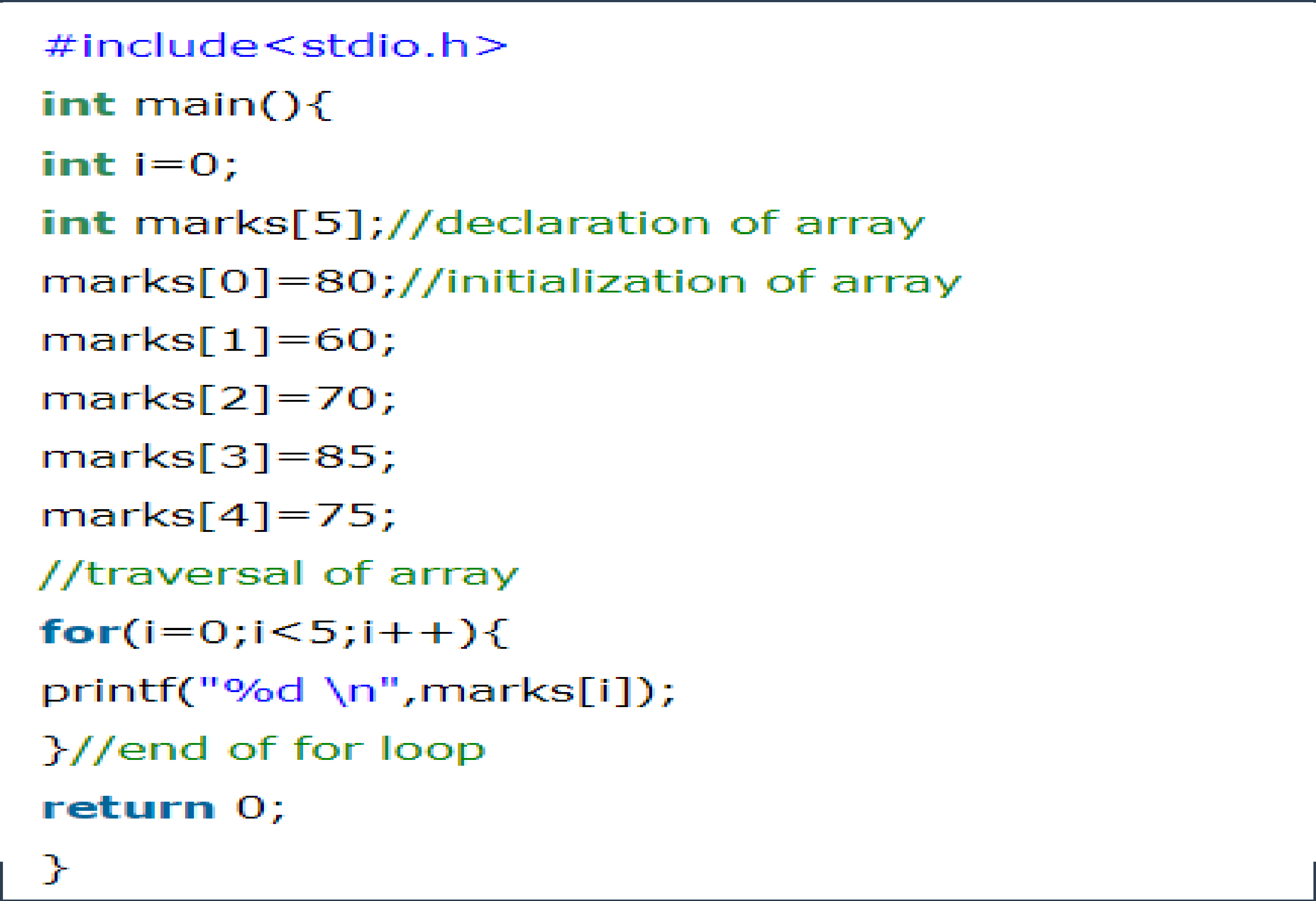
```
int balance[5] = {10, 20, 30, 40, 50};
```

Note: The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ] .

- In such case, there is no requirement to define the size. So it may also be written as the following code.

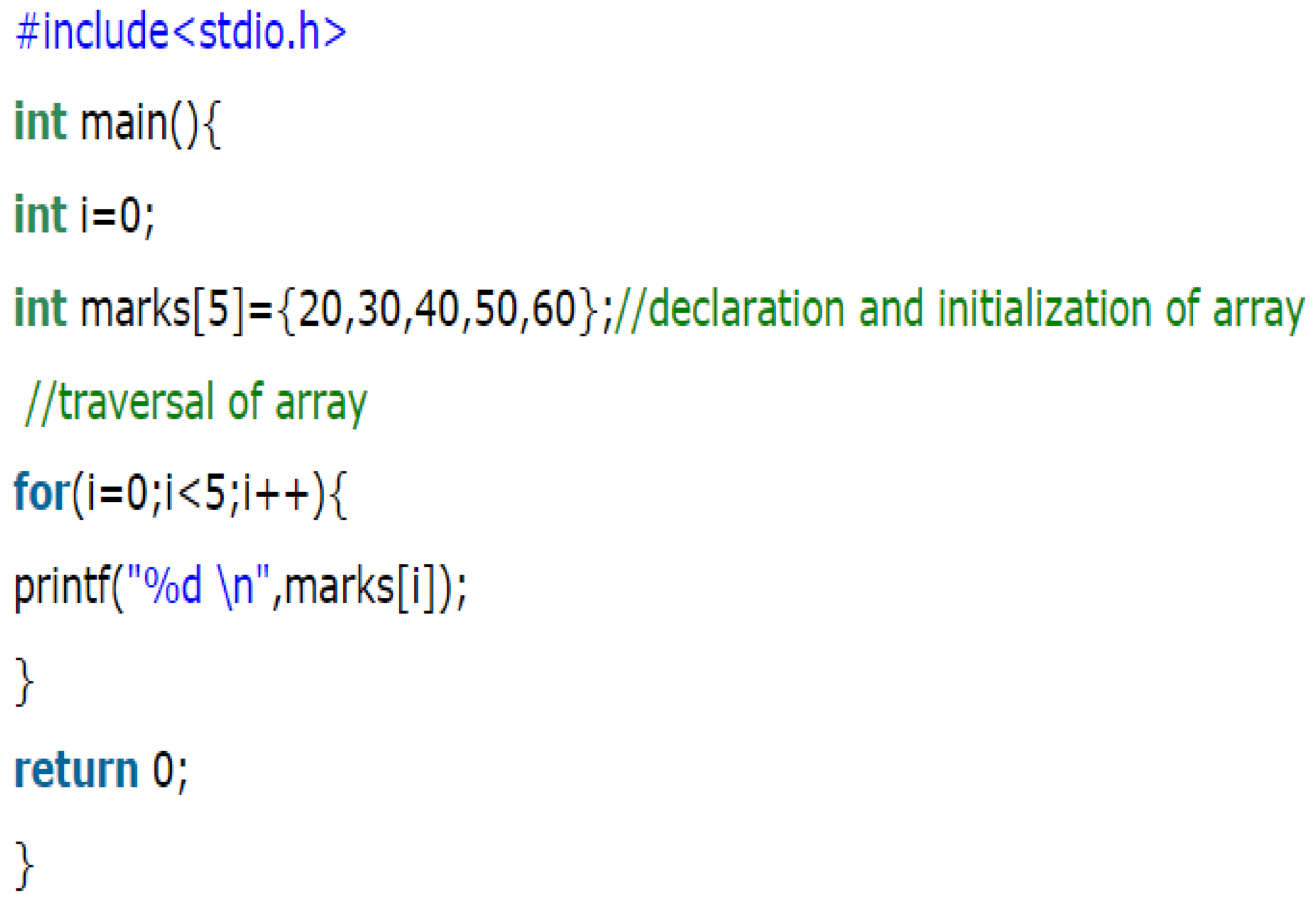
```
int balance[]={20,30,40,50,60};
```

C array example



```c
#include<stdio.h>
int main(){
int i=0;
int marks[5];//declaration of array
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}//end of for loop
return 0;
}
```

C array example



```c
#include<stdio.h>
int main(){
int i=0;
int marks[5]={20,30,40,50,60};//declaration and initialization of array
 //traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}
return 0;
}
```

Types of Array

- There are two types of Array:

  1. **Single Dimensional Array**
  2. **Multi Dimensional Array**

Single Dimensional Array

- A one-dimensional array (or single dimension array) is a type of linear array. Accessing its elements involves a single subscript which can either represent a row or column index.
- **Syntax:**

```
datatype name_of_array [size];
```

- **Example:**

```
int a[10];
```

Multi Dimensional Array

- C supports multidimensional arrays. The simplest form of the multidimensional array is the two- dimensional array.
- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration −
- **Syntax:**

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

- **Example:** the following declaration creates a three dimensional integer array −

    int threedim[2][4][3];

- **Note: the array can hold 24 elements.**

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 −
- **Syntax:**

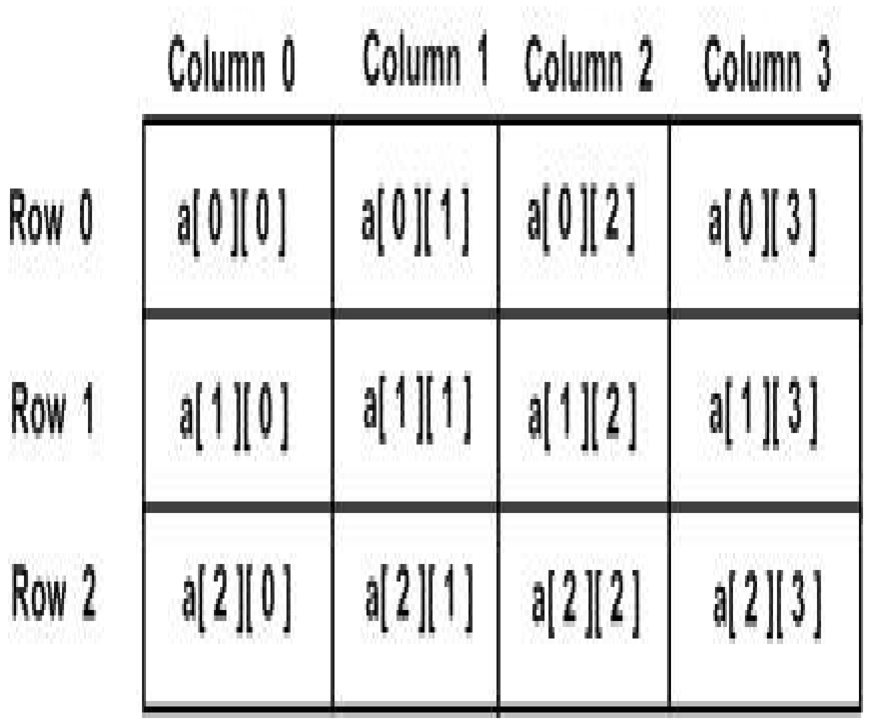
```
type arrayName [ x ][ y ];
```

- 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.

Two Dimensional Array

- A two-dimensional array a, which contains three rows and four columns can be shown as follows-
- **Example:**

  int a[3][4];
- **Note:** the array can hold 12 elements.
- Thus, 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(indexs) that uniquely identify each element in 'a'.



	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]

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.
- **Example:**

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

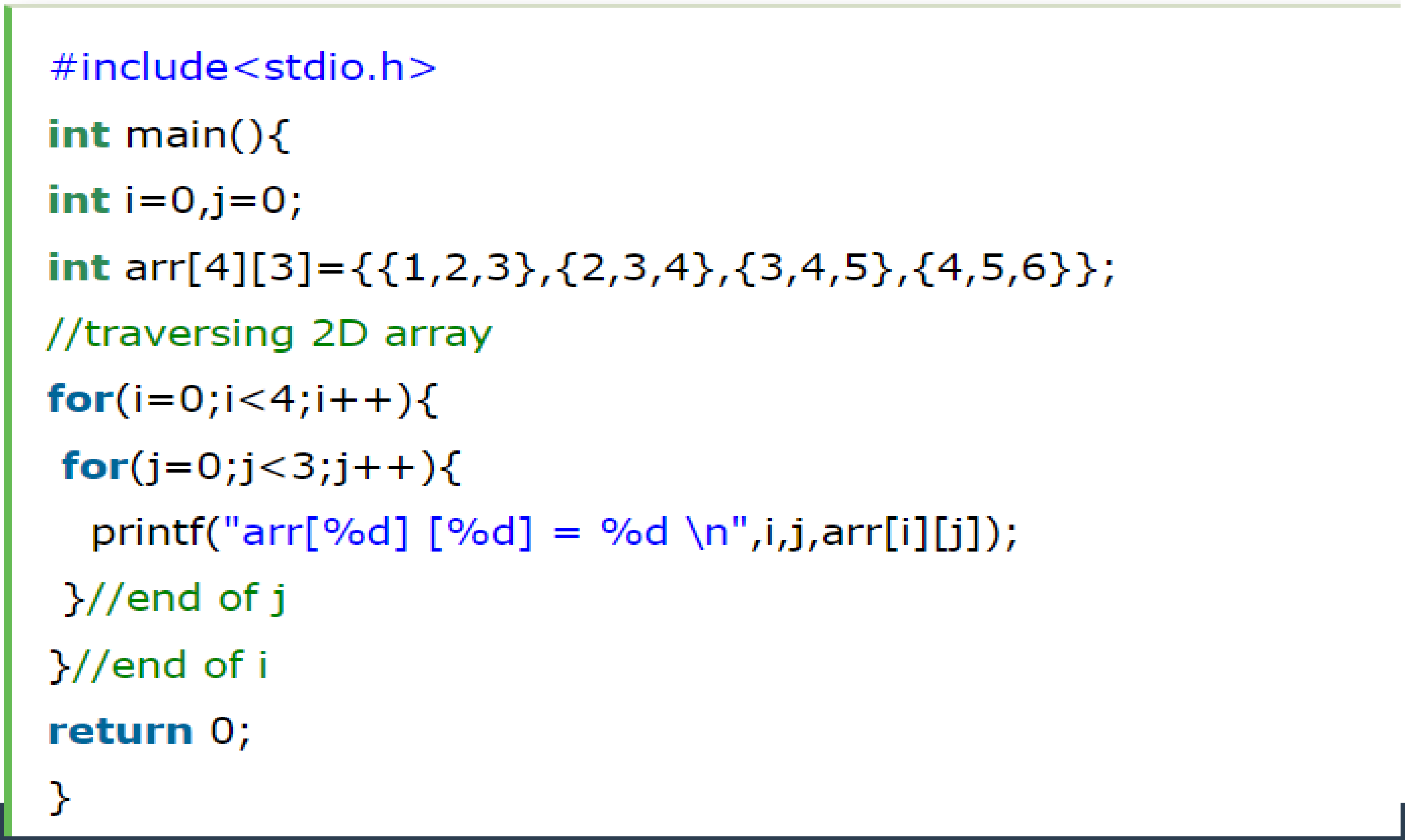
Accessing Two-Dimensional Array Elements

- 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 −
- **Example:**

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

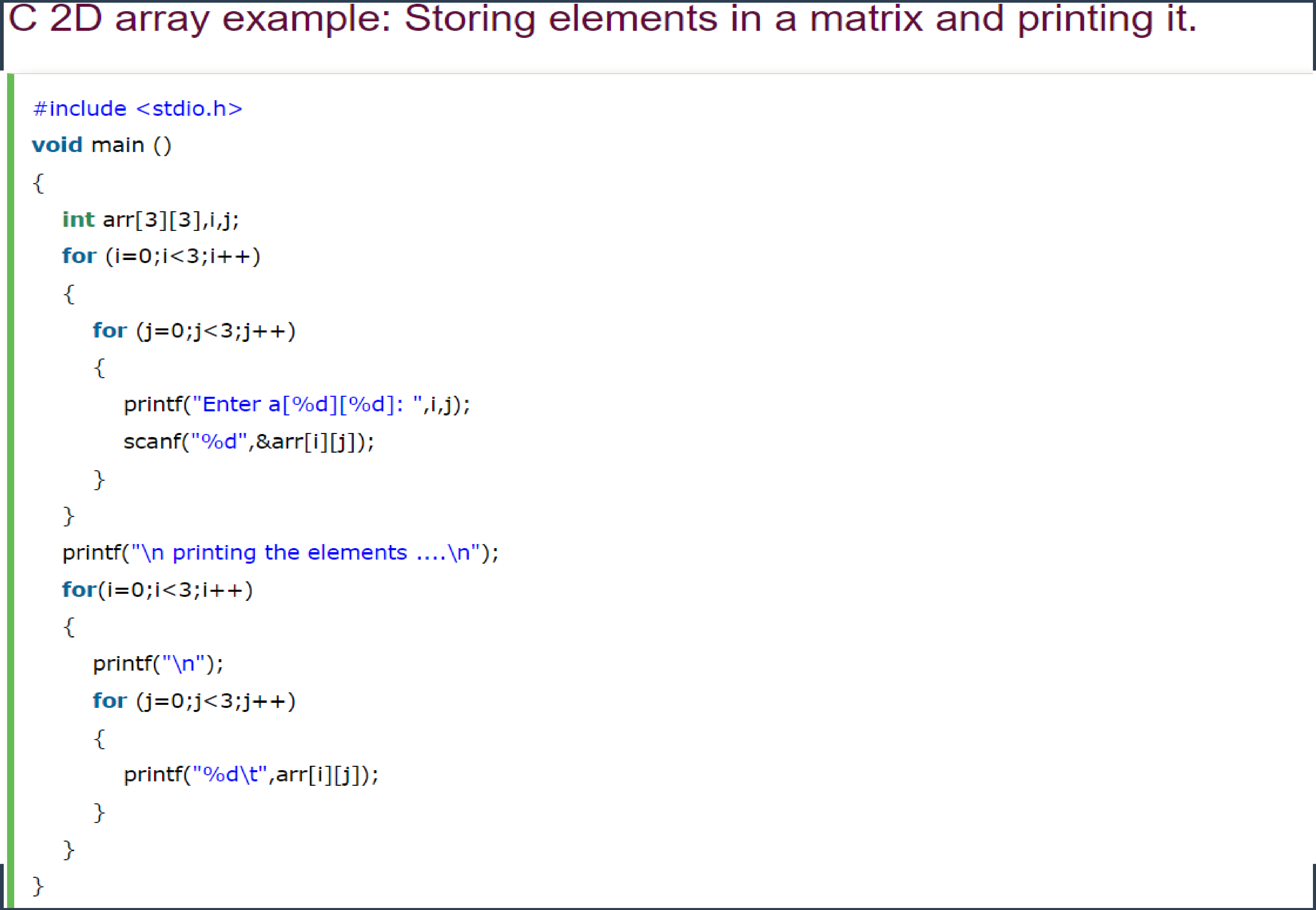
Two Dimensional Array

Two-dimensional array example in C



```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Two Dimensional Array



C 2D array example: Storing elements in a matrix and printing it.

```
#include <stdio.h>
void main ()
{
   int arr[3][3],i,j;
   for (i=0;i<3;i++)
   {
      for (j=0;j<3;j++)
      {
         printf("Enter a[%d][%d]: ",i,j);
         scanf("%d",&arr[i][j]);
      }
   }
   printf("\n printing the elements ....\n");
   for(i=0;i<3;i++)
   {
      printf("\n");
      for (j=0;j<3;j++)
      {
         printf("%d\t",arr[i][j]);
      }
   }
}
```

Storage Classes

- Storage class in C decides the part of storage to allocate memory for a variable, it also determines the scope of a variable.
- All variables defined in a C program get some physical location in memory where variable's value is stored.
- Memory and CPU registers are types of memory locations where a variable's value can be stored.
- **Four different storage classes in a C program :**
- – Auto
- – Extern
- – Static
- – Register

Auto Storage Class

- **Auto Storage Class**

  The auto storage class is the default storage class

  for all local variables.

```
{
    int mount;
    auto int month;
}
```

Extern Storage Class

- **Extern Storage Class**
- The extern storage class is used to give a reference of a global variable that is visible to ALL the program files.
- When you have multiple files and you define a global variable or function, which will also be used in other files, then extern will be used in another file to provide the reference of defined variable or function.
- **extern int count;**

Register Storage Class

- **Register Storage Class**

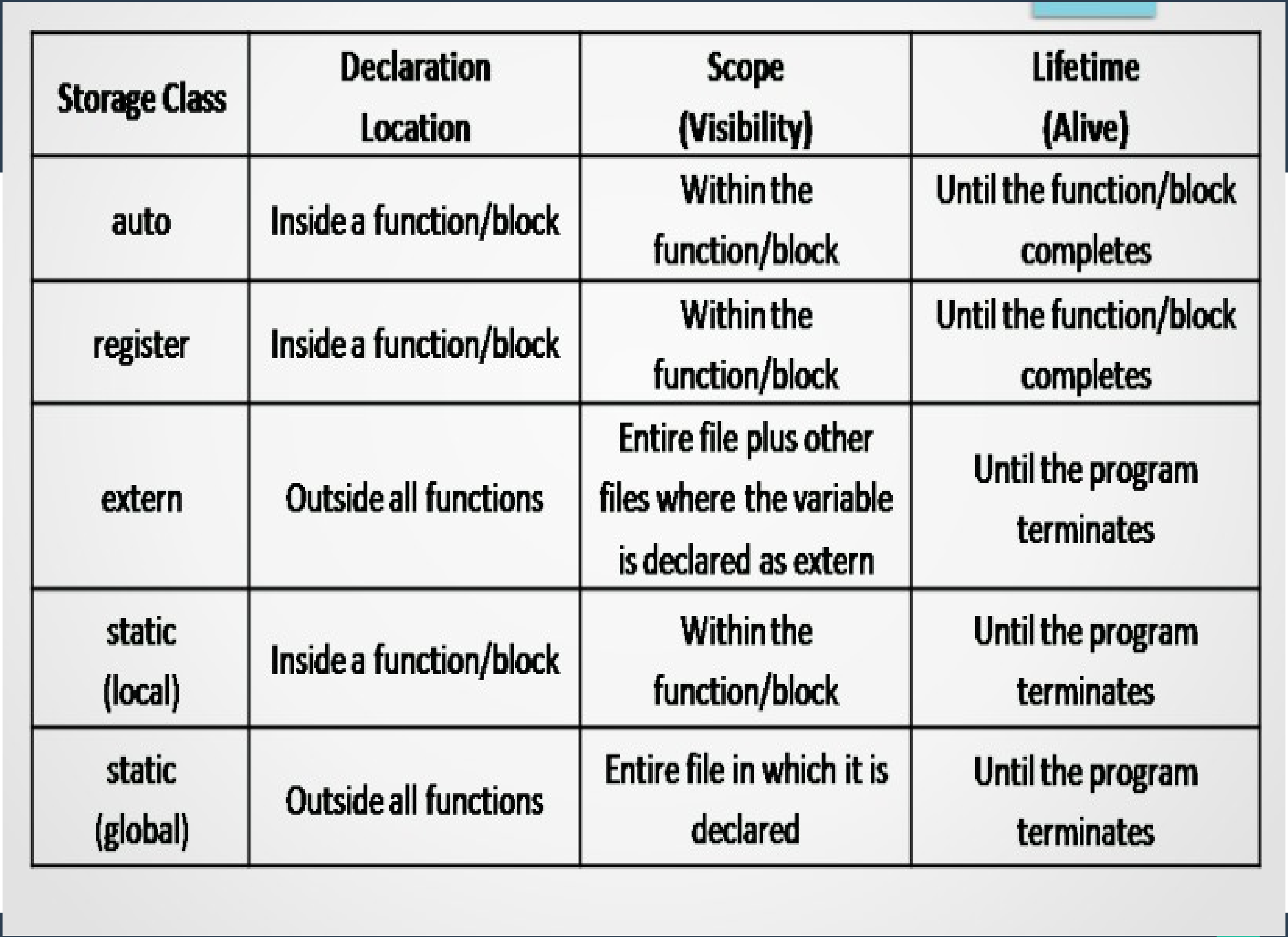
  The register storage class is used to define local variables that should be stored in a register instead of RAM.

  ```
  {
      register int miles;
  }
  ```

  The register should only be used for variables that require quick access such as counters.

Static Storage Class

- **Static Storage Class**
- The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program.
- Making local variables static allows them to maintain their values between function calls.
- In C programming, when static is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.
- static int count = 5;



Storage Class	Declaration Location	Scope (Visibility)	Lifetime (Alive)
auto	Inside a function/block	Within the function/block	Until the function/block completes
register	Inside a function/block	Within the function/block	Until the function/block completes
extern	Outside all functions	Entire file plus other files where the variable is declared as extern	Until the program terminates
static (local)	Inside a function/block	Within the function/block	Until the program terminates
static (global)	Outside all functions	Entire file in which it is declared	Until the program terminates

THANK YOU