Unit 2 Arrays and Pointers

Storage classes: Automatic, External, Static, Register, Scope and lifetime of variables, Arrays and pointers and corresponding operators, Pointer arithmetic, Programs using arrays and pointers like sum, average, minimum, maximum of an array of numbers, Add and delete an element of an array, Merge two sorted arrays, String manipulation programs like addition, subtraction, multiplication and their combinations, Sum of rows, columns, and diagonal elements of a matrix, Transpose of a matrix, Linear search, binary search. Selection sort and bubble sort.

Storage classes in C:

A storage class defines the scope (visibility) and life-time of variables and functions within a C Program. They precede the type that they modify. There are**four** different storage classes in a C program −

- auto
- register
- static
- extern

auto: The auto storage class is the default storage class for all local variables.

Example:

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

The example above defines two variables with in the same storage class.

'auto' can only be used within functions, i.e., local variables.

register: The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the

Example:

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

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register.

static: This storage class is used to declare **static** variables which are popularly used while writing programs in C language. Static variables have a property of preserving their value even after they are out of their scope. Hence, static variables preserve the value of their last use in their scope. So we can say that they are initialized only once and exist till the termination of the program.

Syntax: storage_class <data_type> var_name;

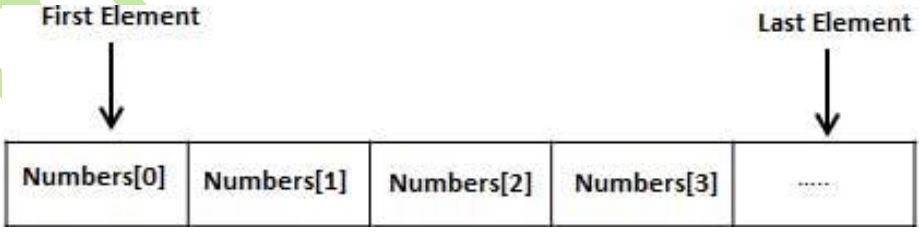
extern: Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten or changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere.

Syntax: storage_class <data_type> var_name;

What are Arrays?

An array is defined as the collection of similar type of data items stored at contiguous memory locations. 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.

It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number. Representation of Array in memory is shown below.



Properties of Arrays:

- Array is Static data Structure, i.e its size is fixed.
- Memory Allocated during Compile time.
- It can hold data belonging to same Data types

Unit 2 Arrays and Pointers

Types of Arrays:

- Single Dimensional Arrays (1D Array)
- Two Dimensional Arrays (2D Array)
- Multi-Dimensional Arrays (3D Array)

Single Dimensional Array: Single or One Dimensional array is used to represent and store data in a linear form. 1D array has only one subscript variable.

Syntax to declare: <data-type><array_name> [size];

Example 1:

```
int arr[3] ;                        // only declaration
char carr[20] = "c4learn" ;         // declaration and initialization
float farr[3] = {12.5,13.5,14.5};// declaration and initialization
```

Two Dimensional Arrays: It is the combination of more than one 1 D arrays. 2D array hastwo subscript variables. 2D arrays are also called as Matrix.

Syntax to declare: <data-type><array_name> [size1] [size2];

size1 indicates the numbers of rows in the matrix

size2 indicates the numbers of columns in the matrix

Example 2:

```
int arr[3][4];                      // only declaration
int a[3][3] = {1,2,3, 5,6,7, 8,9,0};


// declaration and initialization
```

Example 3: WAP in C to display the sum and average of the elements of an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
        float i, a[10], sum=0, avg=0;
        printf("Enter 10 numbers in the Array");
        for(i=0; i<10; i++)
        {
                scanf("%f", &a[i]):
                sum=sum+a[i];
        }
        avg=sum/10;
        printf("\n The SUM is=%f", sum);
        printf("\n The Average is=%f", avg);
        getch();
}
```

Example 4: Addition of Two Matrices using 2D arrays:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int r, c, a[10][10], b[10][10], sum[10][10], i, j;
        printf("Enter the number of rows ");
        scanf("%d", &r);
        printf("Enter the number of columns ");
        scanf("%d", &c);
        printf("Enter elements of 1st matrix:");
        for (i = 0; i < r; ++i)
                for (j = 0; j < c; ++j)
                {
                        scanf("%d", &a[i][j]);
                }

        printf("Enter elements of 2nd matrix:\n");
        for (i = 0; i < r; ++i)
                for (j = 0; j < c; ++j)
                {
                  scanf("%d", &b[i][j]);
                }

        // matrices addition
        for (i = 0; i < r; ++i)
                for (j = 0; j < c; ++j)
                {
                sum[i][j] = a[i][j] + b[i][j];
                }

        // Display final Matrix
        printf("\nSum of two matrices: \n");
        for (i = 0; i < r; ++i)
                for (j = 0; j < c; ++j)
                {
                printf("%d   ", sum[i][j]);
                if (j == c - 1)
                        {
                        printf("\n\n");
                }
                }
        getch();
}
```

Unit 2 Arrays and Pointers

What is a pointer?

A pointer in C language is a variable that stores/points the address of another variable. A pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Pointer variable declaration Syntax: data_type *var_name;

Example: int *p; char *p;

Where, * is used to denote that “p” is pointer variable and not a normal variable.

The unary operator “&” gives the address of a variable".

A simple example is given below.

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int c = 5;
        int *p = &c;
        printf("%d", *p);          //Output : 5
        getch();
}
```

Pointer arithmetic: Following arithmetic operations are possible on pointers in C language:

- Increment and decrement `++` and `--`
- Addition and Subtraction `+` and `-`
- Comparison `<`, `>`, `<=`, `>=`, `==`, `!=`

Incrementing Pointers:

int* i;

i++;

In the above case, pointer will be of 2 bytes. And when we increment it, it will increment by 2 bytes because int is also of 2 bytes.

Again for float variables,

float* i;

i++;

In this case, size of pointer is still 2 bytes (for a 16 bit PC) initially. But now, when we increment it, it will increment by 4 bytes because float datatype is of 4 bytes.

Decrementing Pointers:

If ip is an integer pointer and suppose address of ip is 1010 then

ip = ip - 2

ip => ip - 2 => 1010 - 2*2 => 1006

Example of Pointer Comparison:

```
void main()
{
        int num = 10;
        int *ptr1 = &num;    // ptr1 points to num
        int *ptr2 = &num;    // ptr2 also points to num
        if(ptr1 == ptr2)
        {
                // program statements
        }
}
```

Benefits of using pointers:

Below we have listed a few benefits of using pointers:

- Pointers are more efficient in handling Arrays and Structures.
- Pointers allow references to function and thereby help in passing of function as arguments to other functions.
- It reduces length of the program and its execution time as well.
- It allows C language to support Dynamic Memory management.