Unit - II: Conditional Branching and Loops, Arrays, Strings, Structures ,Unions

Syllabus

Conditional Branching and Loops: Writing and evaluation of conditionals and consequent branching with if, if-else, switch-case, ternary operator, goto, Iteration with for, while, do- while loops

Arrays: one and two dimensional arrays, creating, accessing and manipulating elements of arrays.

Strings: Introduction to strings, handling strings as array of characters, basic string functions available in C (strlen, strcat, strcpy, strstr etc.), arrays of strings

Structures: Defining structures, initializing structures, **unions**, Array of structures

Enumeration data type

CONTROL STATEMENTS

- When we run a program, the statements are executed in sequence order in which they appear in the program.
- Also each statement is executed only once.
- But in many cases we may need a statement or a set of statements to be executed a fixed no of times or until a condition is satisfied.
- Also we may want to skip some statements based on testing a condition.
  For all these we use control statements.

Control statements are of two types –

1. Branching (Decision making) (selection) statement
2. Looping statement(iterative)

Branching (Decision making) statement

- When we need to execute a block of statements only when a given condition is true then we use decision making statement

Decision-making by supporting the following statements,

1. if statement
2. switch statement

1. if statement

if statement having 4 ways

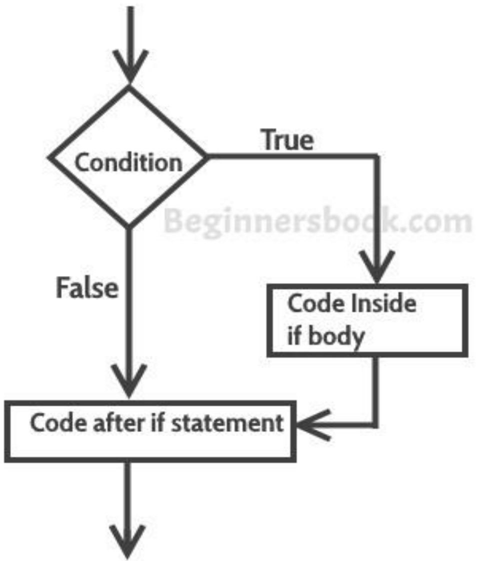
1. Simple if
2. if..else,
3. nested if..else
4. else..if. ladder

simple if statement:

If the condition/expression is "true" statement inside block will be executed,

if condition is "false" then statement_inside block will skipped i.e not executed, statement_outside block excuted.

```
if(expression)
{
                                                    Statement_inside;
}
                                                    Statement_outside;
```



```
#include <stdio.h>
void main()
{
   int x = 20;
   int y = 22;
   if (x<y)
   {
      printf("Variable x is less than y");
   }
   printf("\n End of program \n");
}
```

```
// Program to display a number if it is negative
#include <stdio.h>
void main()
{
   int n;
   printf("\n Enter an integer:\n ");
   scanf("%d", &n);
      if (n< 0) // true if number is less than 0
                  {
                  printf("You entered %d is negative \n", n);
                  }
   printf("End of statement\n");
}
```

output

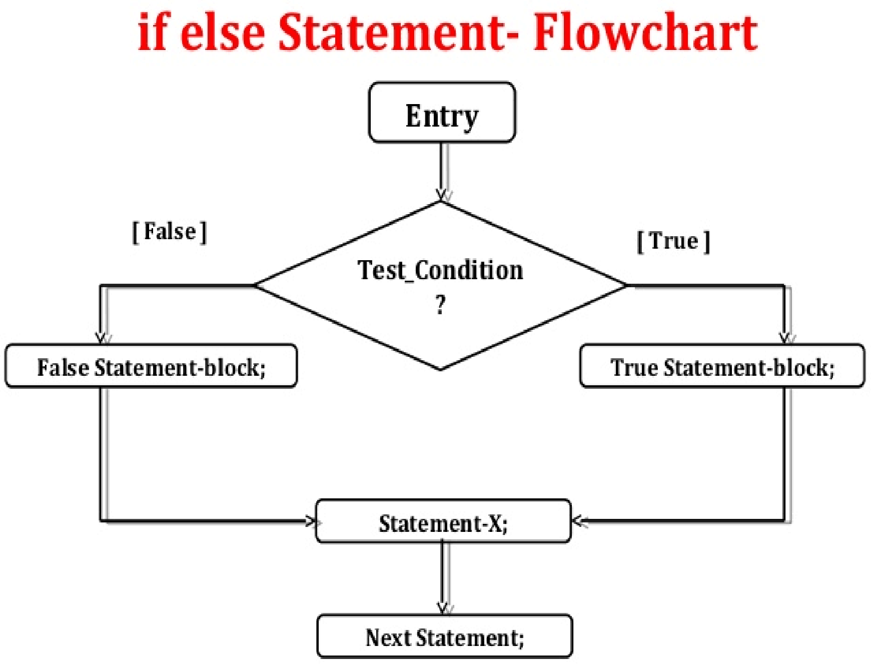
```
Enter an integer: -2
You entered -2 is negative
End of statement
```

2. <u>if...else statement</u>

<u>Syntax:</u>

```
if(expression)
{
   statement block1;
}
else
{
   statement block2;
}
```

If the expression is true, the statement-block1 is executed, else statement-block1 is skipped and statement-block2 is executed.



Example 1: Write a program to check whether the given number is odd or even.

```
#include<stdio.h>
void main()
{
    int n;
```

```
     printf("Enter any Number :\n ");
     scanf("%d", &n);
     if(n%2 == 0)
            {
          printf("%d is Even Number", n);
                         }
            else
            {
           printf("%d is Odd Number", n);
            }
Printf("\n End of program \n");
}
```

Output:

```
Enter any Number : 60
60 is Even Number
End of program
```

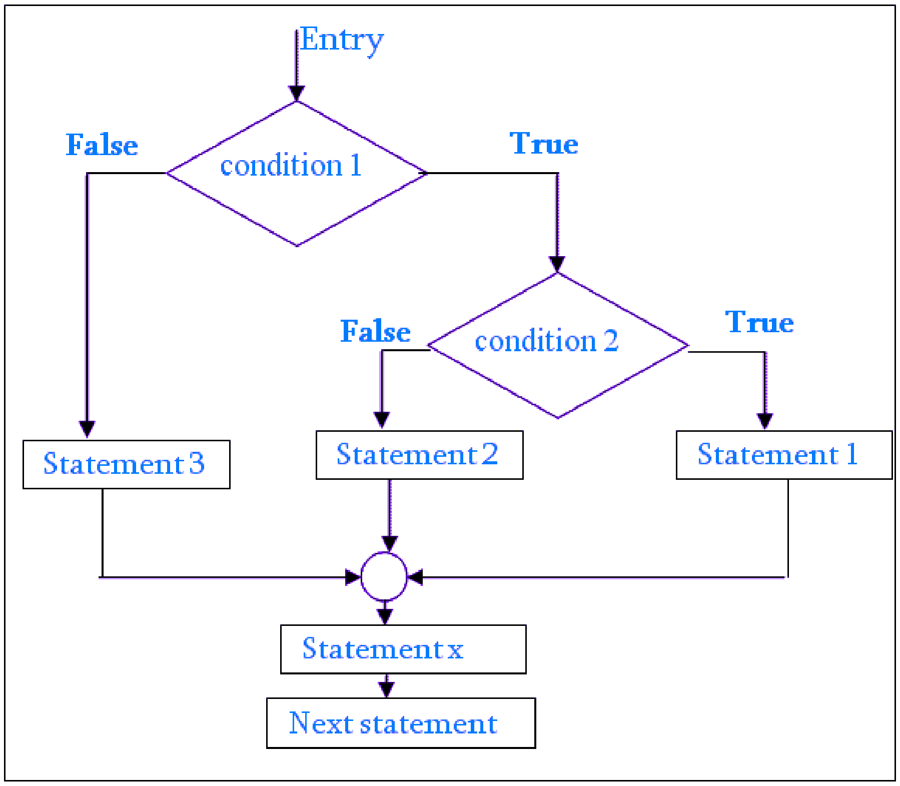
3. **Nested if....else statement**

- Condition in condition is called nested if ..else statement

Syntax:

```
if( expression1 )
{
   if( expression2 )
   {
      statement block1;
   }
   else
   {
      statement block2;
   }
}
else
{
   statement block3;
}
```

if expression1 is false then statement-block3 will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if expression 2 is true the statement-block1 is executed otherwise statement-block2 is executed.



Flow chart for

Nested if....else statement

Example 1: Write a program to find greatest number among the 3 numbers.

```
#include <stdio.h>
void main( )
{
   int a, b, c;
   printf("Enter 3 numbers...:\n ");
   scanf("%d%d%d",&a, &b, &c);
   if(a > b)
   {
      if(a > c)
      {
         printf(" %d is the greatest\n",a);
      }
      else
      {
         printf("%d is the greatest\n",c);
      }
```

```
    }
    else
    {
        if(b > c)
        {
            printf("%d is the greatest\n",b);
        }
        else
        {
            printf("%d is the greatest \n",c);
        }
    }
Printf("\n End of prog");
}// main
```

Output:

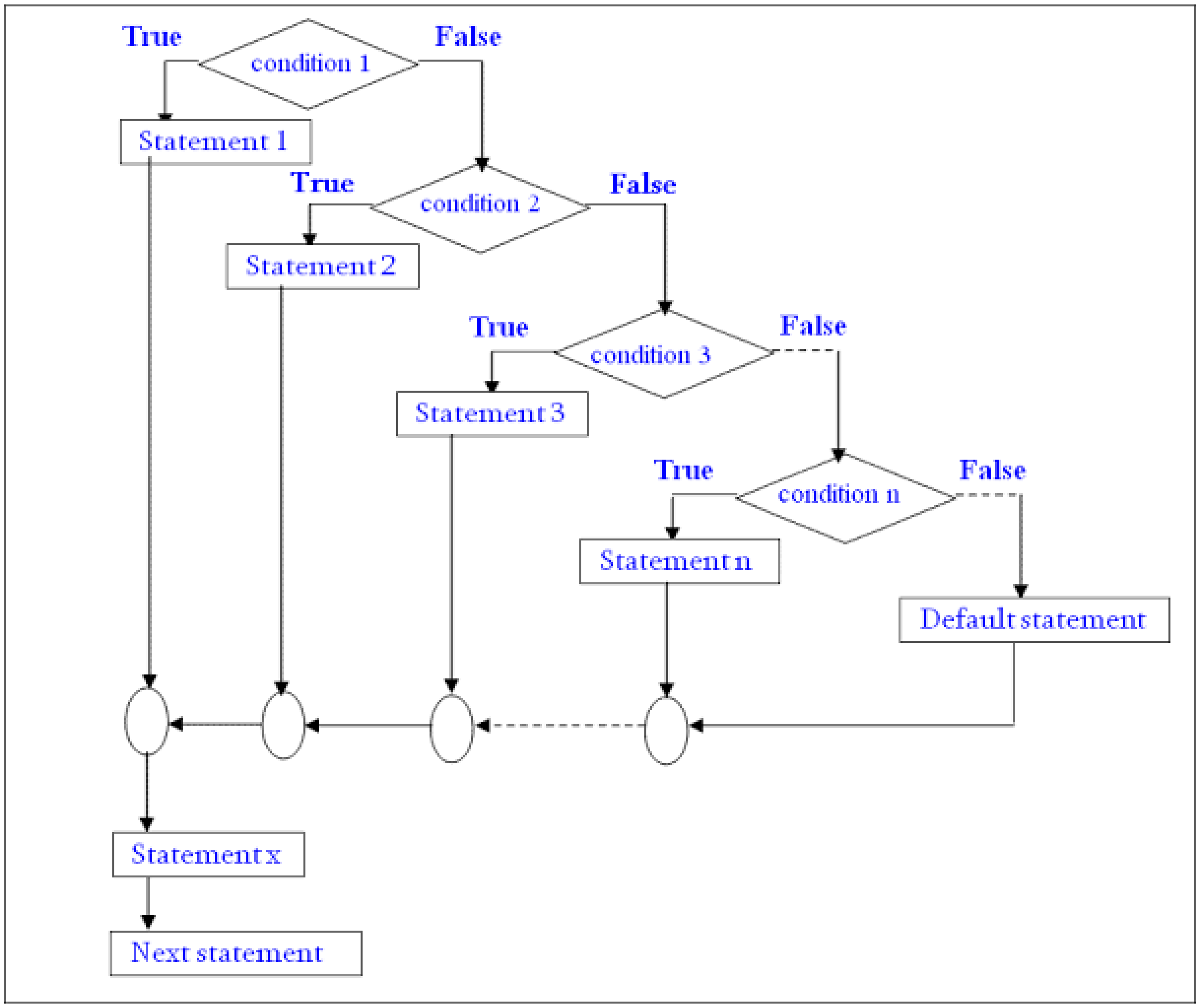
```
Enter 3 numbers...:
12
3
5

 12                              is the greatest
```

4. **else if ladder**

```
if(expression1)
{
    statement block1;
            }
            else if(expression2)
            {
             statement block2;
            }
            else if(expression3 )
            {
              statement block3;
            }
            else
            {
    default statement;
            }
Statement_Next;
```

The expression is tested from the top (of the ladder) downwards. As soon as a true condition is found, the statement associated with it is executed.

Flow chart for else if ladde



Example 1 : WAP a program to get the percentage and print the division;
Conditions:

if percentage >=75 , Division = Distinction
if percentage<75 and >=60, Division = First
if percentage < 60 and >=40, Division = Second
if percentage < 40 and >=35, Division = Third
Otherwise fail ..

```
#include<stdio.h>
void main()
{
    int per;
    printf("\nEnter your percentage : \n");
    scanf("%d",&per);
    if(per>=75)
        printf("Division = Distinction\n");
    else if(per < 75 && per>=60)
        printf("Division = First\n");
    else if(per<60 && per>=45)
        printf("Division = Second\n");
    else if(per<45 && per >=35)
        printf("Division = Third\n");
    else
        printf("Division = Fail ...\n ");

    printf(“\n End of program\n”);
}
```

Output:

```
Enter your percentage: 68
Division = First
End of program
```

SWITCH

Syntax:

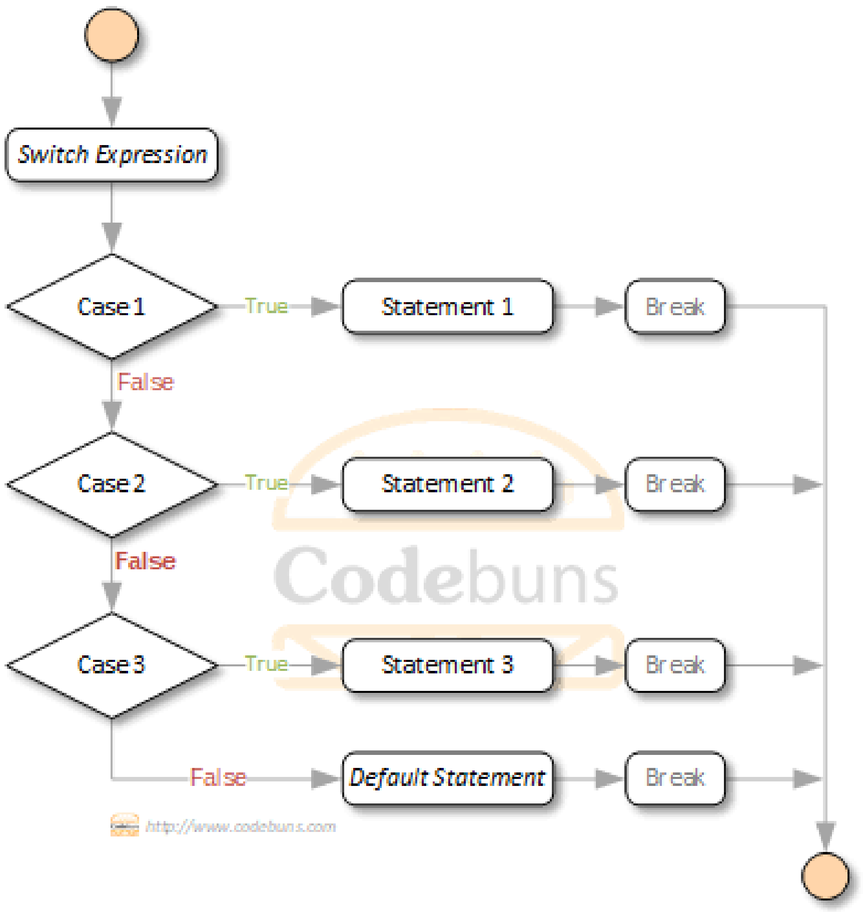
```
switch (expression)
{
   case  label1:
      statements1
      break;
    case  label2:
       statements2
      break;
  case  label3:
       statements3
      break;
    .
    .
    default:
      default statements;
            break;
}
```

switch statement :

- The switch-case statement is a better way of writing a program with series of if-else ladder occurs.
- Switch is called Multi-Decision statement.

The switch statement successively tests the value of an expression against a list of case values, if any match is found, the statement associated with that case are executed.

Note: if label is non-integer it has to enclosed with in single quotes (ex: case ‘a’, case ‘+’, case 1, case 2…)



Example:

Simple Calculator

```
// Program to create a simple calculator
#include <stdio.h>
void  main()
 {
    char operator;
    float n1, n2;
    printf("Enter two operands: \n ");
    scanf("%lf %lf",&n1, &n2);
    printf("Enter an operator (+, -, *, /): \n ");
    scanf("%c", &operator);
    switch(operator)
    {
        case '+':
            printf("%f + %f = %f\n",n1, n2, n1+n2);
            break;
        case '-':
            printf("%f - %f = %f\n",n1, n2, n1-n2);
            break;
        case '*':
            printf("%f * %f = %f\n",n1, n2, n1*n2);
            break;
        case '/':
            printf("%f / %f = %f\n",n1, n2, n1/n2);
            break;
        // operator doesn't match any case constant +, -, *, /
        default:
            printf("Error! operator is not correct");
            break;
    }
    printf("\n end of program\n");
}
```

Output:

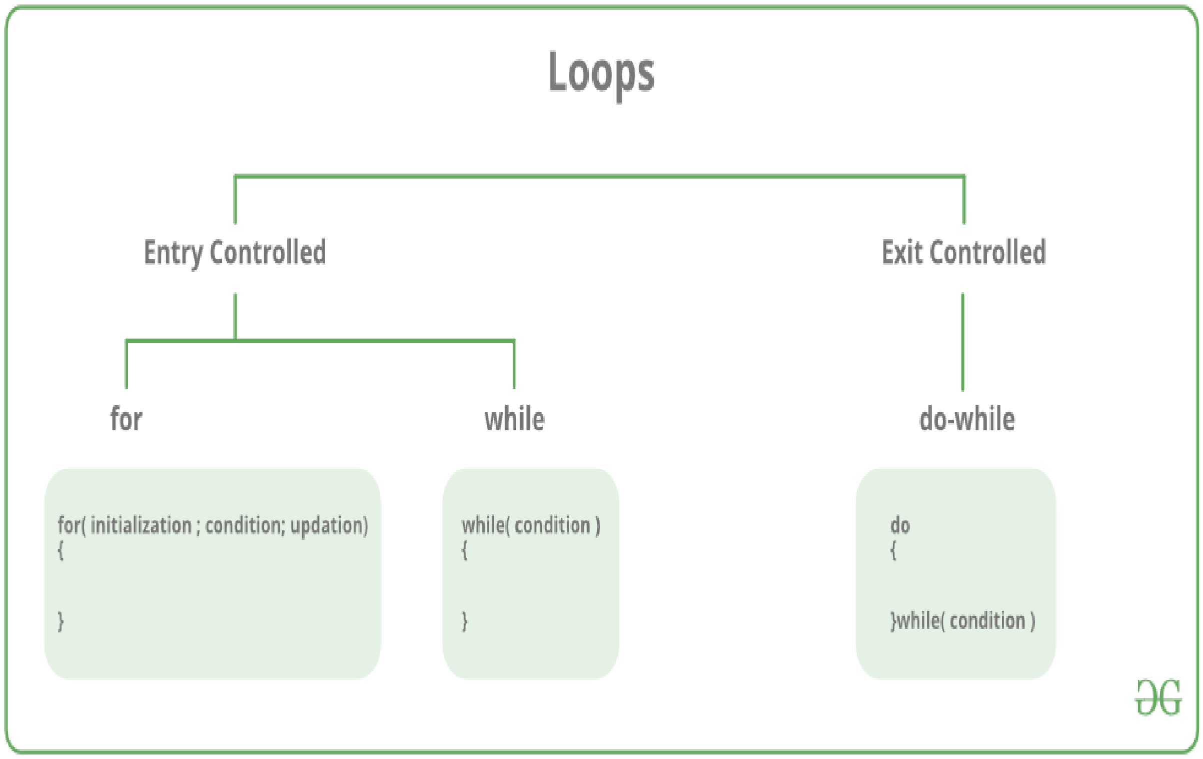
```
Enter two operands:
32.5
12.4
Enter an operator (+, -, *,/):
-
32.5 - 12.4 = 20.1
End of program
```

LOOP OR ITERATIONS STATEMENT

In looping, the set statement are executes until the condition becomes false.

There are mainly two types of loops:

1. **Entry Controlled loops**: In this type of loops the test condition is tested before entering the loop body. **For Loop** and **While Loop** are entry controlled loops.
2. **Exit Controlled Loops**: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. **do – while loop** is exit controlled loop.



Types of Loops

There are 3 types of Loop in C language, namely:

1. while loop
2. for loop
3. do while loop

<u>while loop</u>

<u>Syntax :</u>

```
while(condition)
{
   statements;

}
```

<u>It is completed in 3 steps.</u>

Variable initialization.(e.g int x = 0;)

Condition (e.g while(x <= 10))

Variable increment or decrement (x++ or x-- or x = x + 2)

<u>Working while loop</u>

In while loop, condition is executed first, If the condition is true the body loop will executed , again check the condition if true the body loop will be executed this process will be continue until the condition false then the loop is terminated,

Note: while loop can be called as an entry control loop

Example: Program to print first 10 natural numbers

```
#include<stdio.h>

void main( )
{
   int x;
   x = 1;
   while(x <= 10)
   {
printf("%d\t", x);  /* do x = x+1, increment x by 1*/
      x++;
   }// while close
}// main close
```

Output:
```
1 2 3 4 5 6 7 8 9 10
```

do…while loop

Syntax:

```
    do{
        statement (s);
    } while(condition);
```

Working do-while loop

In do-while loop, the body of the loop is executed first without testing conditions. At the end of the loop, test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop is terminated,

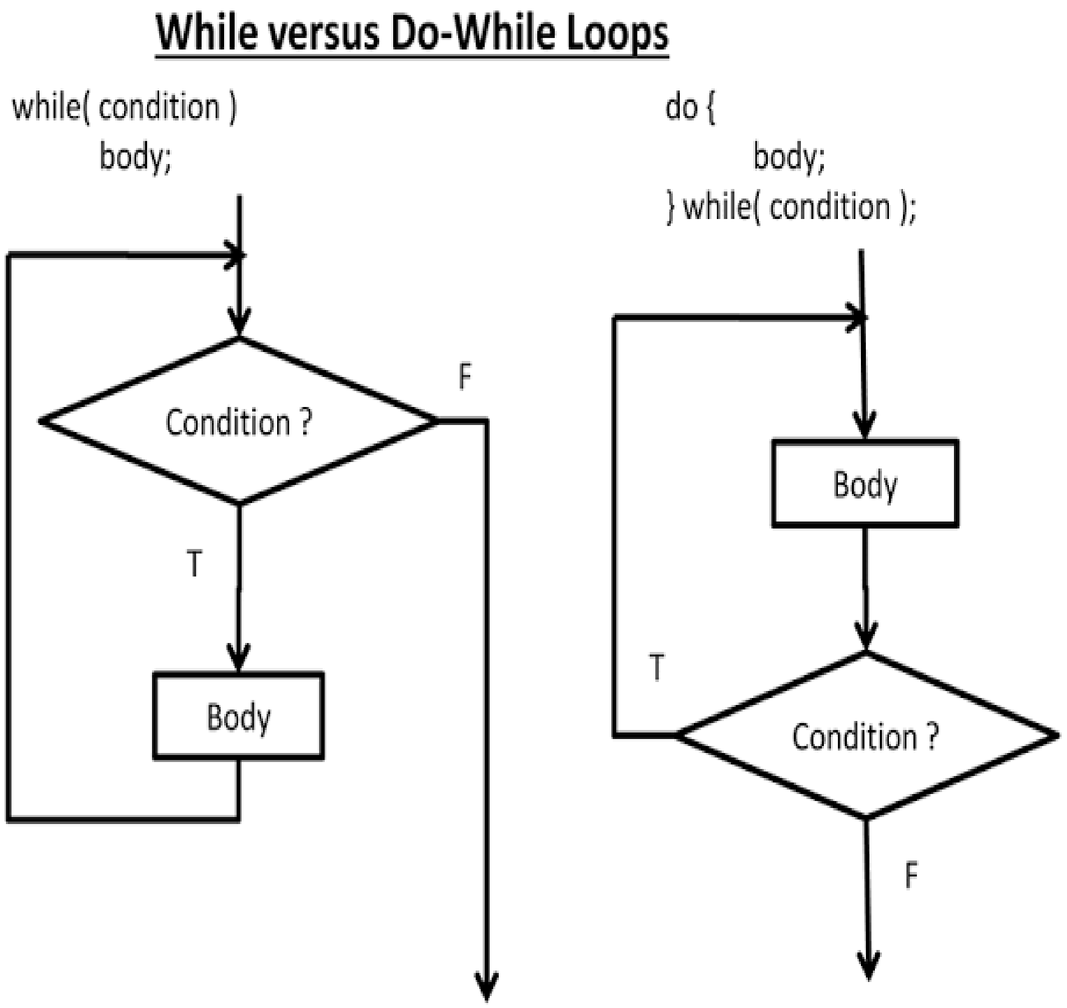
Note: do..While loop can be called as an exit control loop.

Example: Program to print first 3 natural numbers

```
#include <stdio.h>
int main()
{
        int j=0;
        do
        {
                printf("Value of variable j is: %d\n", j);
                j++;
        }while (j<=3);
        return 0;
}
```

Output

```
value of variable j is: 0
Value of variable j is: 1
Value of variable j is: 2
Value of variable j is: 3
```



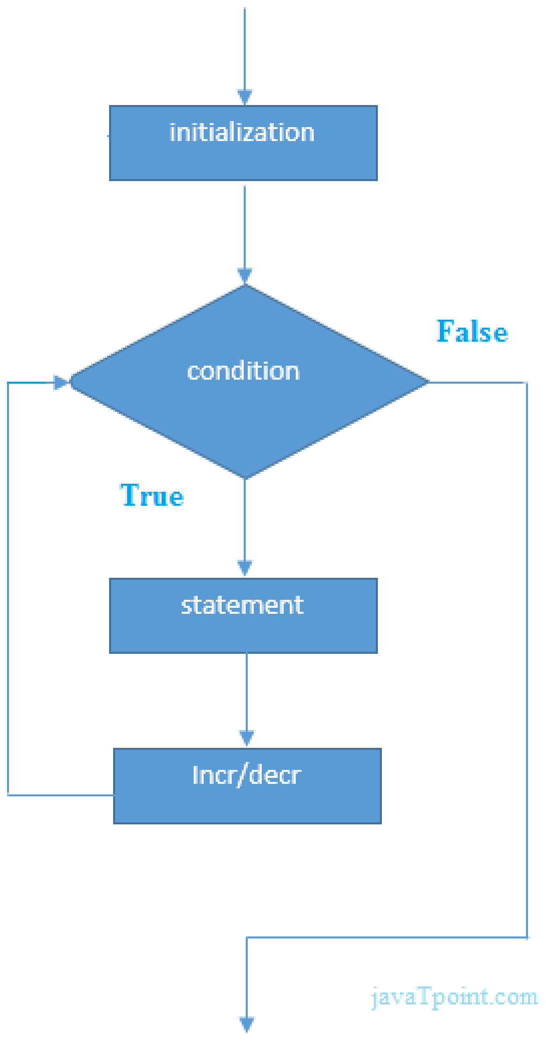
For loop

Syntax of for loop:

```
for (initialization; condition test; increment or decrement)
{
      Statements // body of loop
}
```

<u>The for loop is executed as follows:</u>

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is true, it executes the for-loop body.
4. Then it evaluates the increment/decrement again check the condition follows from step 2.
5. When the condition expression becomes false, it exits the loop.



<u>Example 1: WAP to ask the integer number n and calculate the sum of all</u> natural numbers from 1 to n.

```
#include<stdio.h>
void main()
{
 int n, i, sum=0;
 printf("Enter any number :\n ");
 scanf("%d",&n);
 for(i=1; i<=n; i++)
{
  sum = sum + i;
 }
 printf("The Sum of Natural Numbers = %d\n ",sum);
}
```

Output

```
Enter any number : 5
The Sum of Natural Numbers = 5050
```

<u>Example 2: WAP to calculate factorial of a given number.</u>

```
#include<stdio.h>
void main()
{
 int n, i, f=1;
 printf("Enter any number :\n ");
 scanf("%d",&n);
 for(i=1; i<=n; i++)
{
  f=f*i;
 }
 printf("The Factorial of %d is %d\n",n, f);
}
```

<u>Output</u>

```
Enter any number : 5
The Factorial of 5 is 120
```

Example 3 : Program to print first 10 natural numbers

```
#include<stdio.h>
void main( )
{
    int x;
    for(x = 1; x <= 10; x++)
    {
        printf("%d\t", x);
    }
}
```

<u>output</u>

```
1 2 3 4 5 6 7 8 9 10
```

<u>/ / C program to illustrate for loop</u>

```
#include <stdio.h>

void main()
{
    int i=0;

    for (; i <= 5; )
    {
        printf( "Hello World\n");
i++;
    }
  printf("\n end");
}
```

<u>Output:</u>

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Arrays in C –(PART-A_)

Definition:

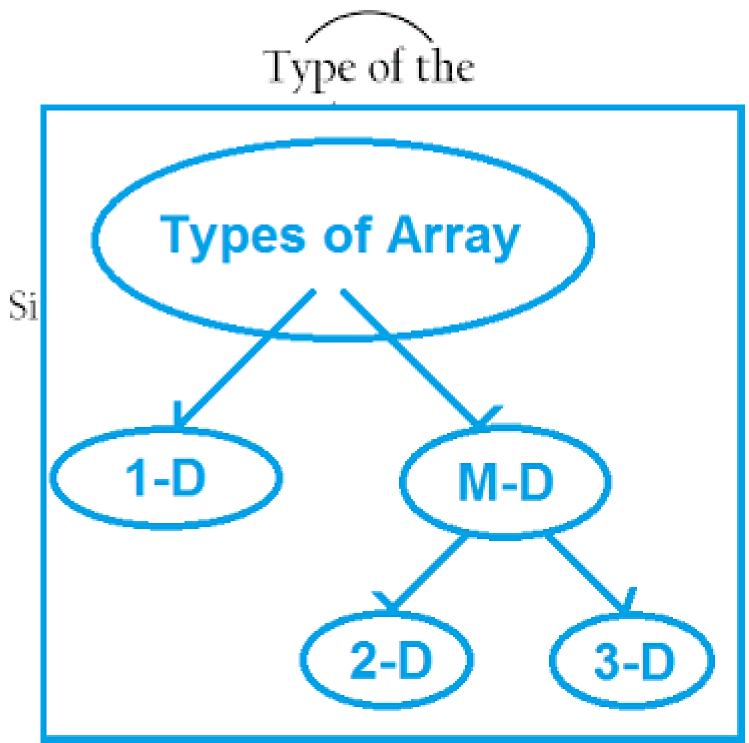
An array is a group (or collection) of similar (homogeneous) data types with one common name.

OR

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

OR

Array is a collection of similar type of elements that share the common name.



TYPES OF C ARRAYS:

There are 2 types of C arrays. They are,

1. One dimensional array
2. Multi dimensional array
    - Two dimensional array
    - Three dimensional array
    - four dimensional array etc…

How to declare an array? (1 –Dimensional)

Syntax to declare array:

```
dataType  arrayName[arraySize];
```

Here

- ➢ Data type any data type i.e int, char, float, double.
- ➢ Arrayname: user defined name
- ➢ Arraysize : must be integer

[]-→ Single   dimension array.

[][]->Two dimensional array.

[][][]->three dimensional array.

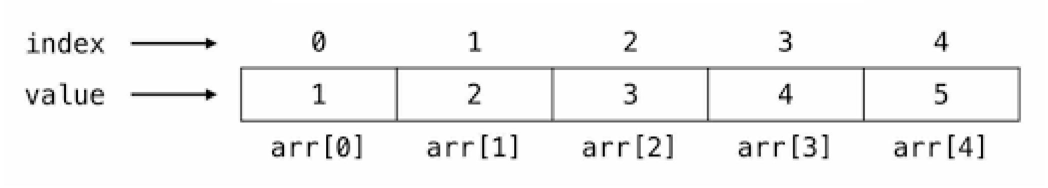
Ex: int   a[10];

float   sum[10];

char   ch[20]; //string

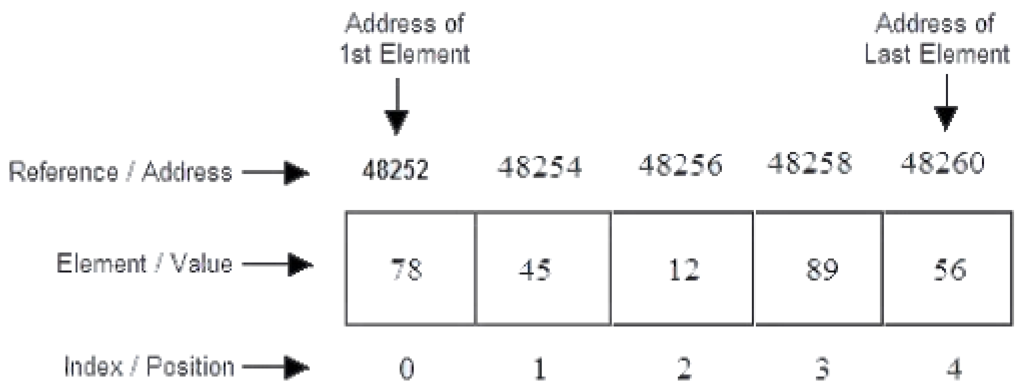
<u>Example:</u>

int a[5];



Ex:

int a[5]={78,45,12,89,56};



Here int is the data type, a is the name of the array and 5 is the size of array. It means array arr can only contain 5 elements of int type.

Note: Index of an array starts from 0 to size-1 i.e first element of arr array will be stored at a[0] address and the last element will occupy a[4]

Initialization of an Array

After an array is declared it must be initialized. Otherwise, it will contain garbage value (any random value).

An array can be initialized at either

- compile time Initialization
- runtime Initialization

Compile time Array initialization

Compile time initialization of array elements is same as ordinary variable initialization.

The general form of initialization of array is,

Syntax:

data-type array-name[size] = { list of values };

```
/* Here are a few examples */

int marks[4]={ 67, 87, 56, 77 };    // full array initialization

float area[5]={ 23.4, 6.8, 5.5 };   // float array initialization

int a[5]={2,3}; //partial array intialzation

int marks[4]={ 67, 87, 56, 77, 59 };    // Compile time error
```

Runtime Array initialization

- An array can also be initialized at runtime using scanf() function.
- This approach is usually used for initializing large arrays, or to initialize arrays with user specified values.

Example,

```
#include<stdio.h>

void main()
{
   int arr[4];
   int i, j;
   printf("Enter array element:\n");
   for(i = 0; i < 4; i++)
   {
      scanf("%d", &arr[i]);    //Run time array initialization
   }
Printf("\n The array elements are:\n")
   for(j = 0; j < 4; j++)
   {
      printf("%d\n", arr[j]);
   }
}
```

Output:

```
Enter array element:
10
20
30
40

The array elements are:
10
20
30
40
```

Example 2: Calculate Average

```
// Program to find the sum and average of n numbers using arrays
#include <stdio.h>
void main()
{
    int marks[50], i, n, sum = 0, avg=0;
    printf("\n Enter the Size of array:\n");
    scanf("%d", &n);
    for(i=0; i<n; ++i)
    {
        printf("Enter number%d:\n",i+1);  // print the index of array
        scanf("%d", &marks[i]); // reading array elements
        sum = sum+marks[i]; // calculate sum


    }
    printf(“\n sum of marks=%d\n”,sum);
    avg = sum/n;
    printf("Average = %d\n”, avg);

}
```

Output:

```
Enter the size of array: 5
Enter number1: 45
Enter number2: 35
Enter number3: 38
Enter number4: 31
Enter number5: 49
sum of marks=198
Average = 39
```

Ex: C program to find max and min of elements in array.

```
#include<stdio.h>
main()
{
```

```
int   a[50];
int  n,i,max,min;
printf(“how many elements”);
scanf(“%d”,&n);

printf(“enter the  number of  elements”);
for(i=0;i<n;i++)
{
scanf(“%d”,&a[i]);
}

max=a[0];
min=a[0];

for(i=1;i<n;i++)
{
if(a[i]>max)
max=a[i];
if(a[i]<min)
min=a[i];
}
printf(“max=%d and min=%d”,max,min);
}
```

Output

```
How many elements: 5
Enter the elements:  50, 40 ,90, 80, 70
Max element is : 90
Min element is : 40
```

Ex: Write a C program to display the array elements in reverse order.

```
#include<stdio.h>
main()
{
 int   a[10],i,n;

printf("enter n value");
scanf("%d",&n);

printf("enter the  array  elements");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}


Printf("display the array elements in reverse order");

for(i=n-1;i>=0;i--)
{
Printf("%d",a[i]);
}

}
```

```
Output:
Enter  n  value: 5
Enter  array elements:   1  2 3  4 5
Display the elements in reverse order:   5 4  3 2 1 .
```

Advantage of C Array

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.

Disadvantage of C Array

1) **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 like LinkedList which we will learn later.

Applications of Arrays in C

In c programming language, arrays are used in wide range of applications. Few of them are as follows...

1. **Arrays are used to Store List of values**

In c programming language, single dimensional arrays are used to store list of values of same data type. In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form.

2. **Arrays are used to Perform Matrix Operations**

We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

3. **Arrays are used to implement Search Algorithms**

We use single dimensional arrays to implement search algorithms like ...

1. **Linear Search**
2. **Binary Search**

4. Arrays are used to implement Sorting Algorithms

We use single dimensional arrays to implement sorting algorithms like.

1. **Insertion Sort**
2. Bubble Sort
3. **Selection Sort**
4. **Quick Sort**
5. Merge Sort, etc.,

5. Arrays are used to implement Data structures

We use single dimensional arrays to implement data structures like...

1. **Stack Using Arrays**
2. **Queue Using Arrays**

6. Arrays are also used to implement CPU Scheduling Algorithms

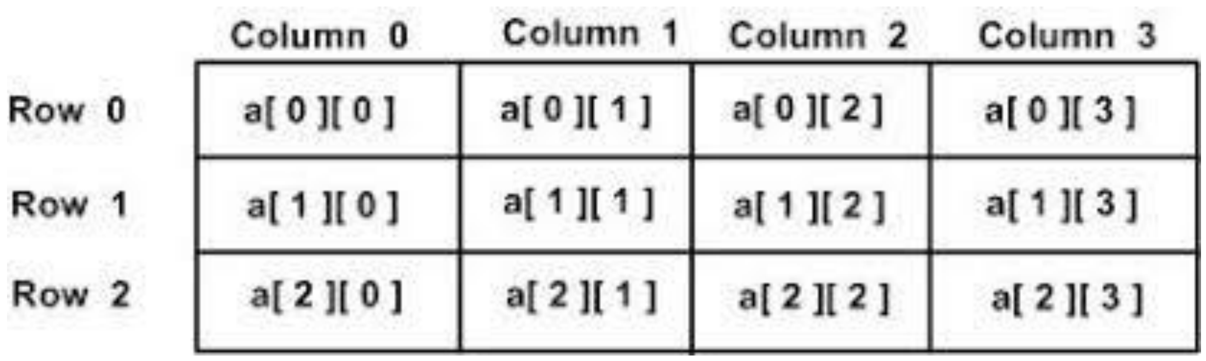
Two dimensional Arrays

- C language supports multidimensional arrays also.
- An array of arrays is known as 2D array.
- The two dimensional (2D) array in C programming is also known as matrix.
- A matrix can be represented as a table of rows and columns.

Two-dimensional arrays **declaration:**

Syntax:
data-type array-name[row-size][column-size];

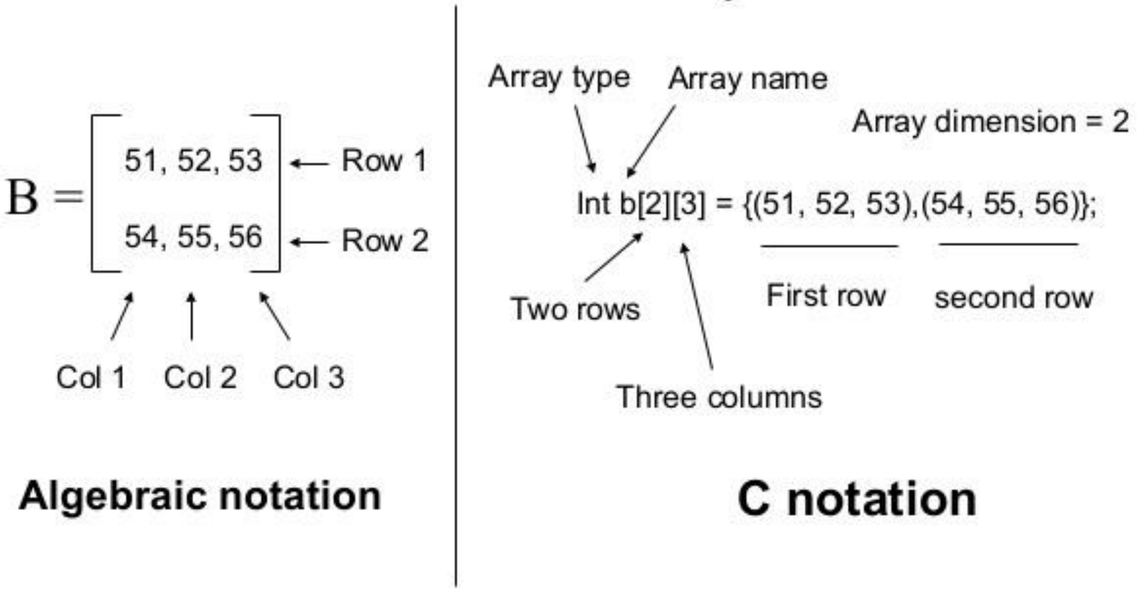
```
/* Example */

int a[3][4];
```

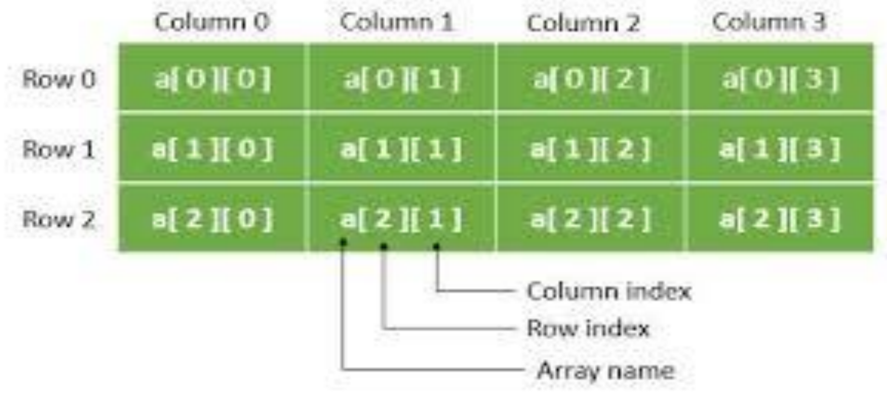


	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Two - Dimensional Arrays

- What is a Two-dimensional array?





An array can also be declared and **<u>initialized together</u>**.

1. Compile Initialization

<u>For example:</u>

data-type array-name[row-size][column-size]

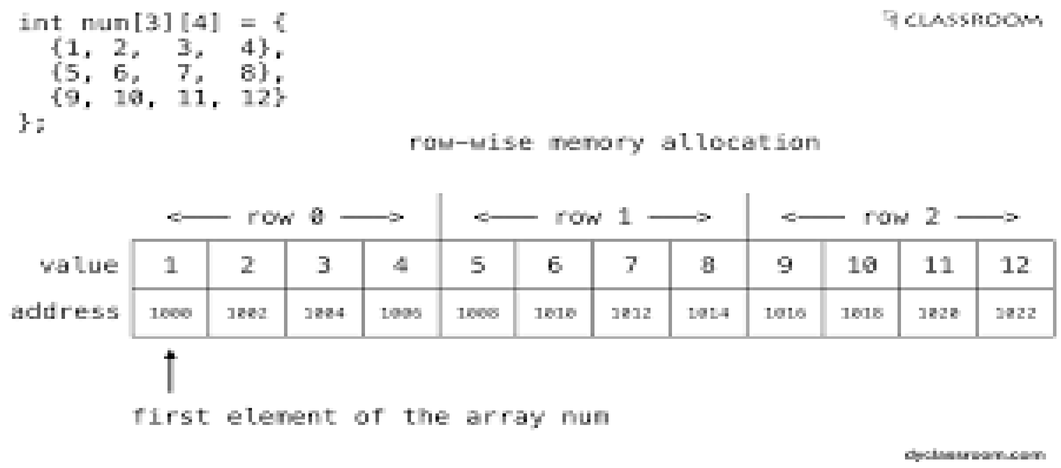
= {{list of elements in row1}, {list of elements in row2},….{list elements in row n}}

int arr[2][3] = {{0,0,0},{1,1,1}};

Note: We have not assigned any row value to our array in the above example.

It means we can initialize any number of rows. But, we must always specify number of columns, else it will give a compile time error. Here, a 2*3 multi-dimensional matrix is created.

Example:



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

Initialization of a 2d array

```
     // Different ways to initialize two-dimensional array
1.  int  c[2][3] = {{1, 3, 0}, {-1, 5, 9}}; // full Initialization
2.  int  c[2][3] = {{1, 3}, {-1, 5, }}; //partial Initialization
3.  int  c[2][3] = {1, 3, 0, -1, 5, 9}; // wrong
```

Runtime initialization of a two dimensional Array

```
#include<stdio.h>
void  main()
{
   int  arr[10][10];
   int  i, j;
   printf("Enter array element:\n");
   for(i = 0; i < 2;i++)
   {
      for(j = 0; j < 3; j++)
      {
         scanf("%d", &arr[i][j]);
      }
   }
   printf("\n The array elements are:\n");
   for(i = 0; i < 2; i++)
   {
      for(j = 0; j < 3; j++)
      {
         printf("%d\t", arr[i][j]);
```

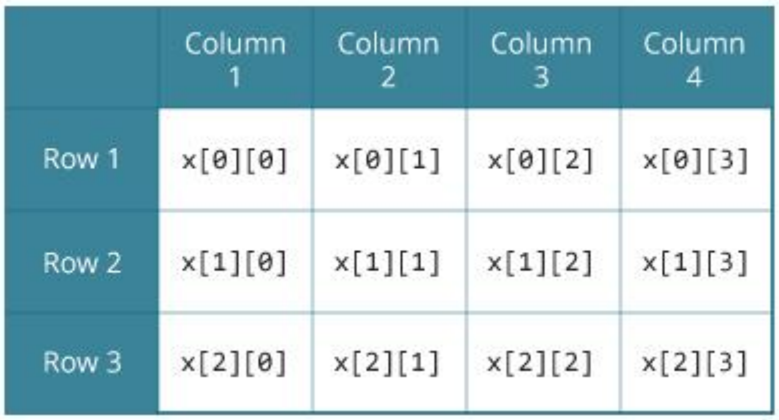
```
        }
            printf("\n");
    }


}
```

```
Enter array element:
1
2
3
4
5
6
The array elements are:
1	2	3
4	5	6
```

In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example,

```
float x[3][4];
```

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.



	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

<u>Initializing a multidimensional array</u>

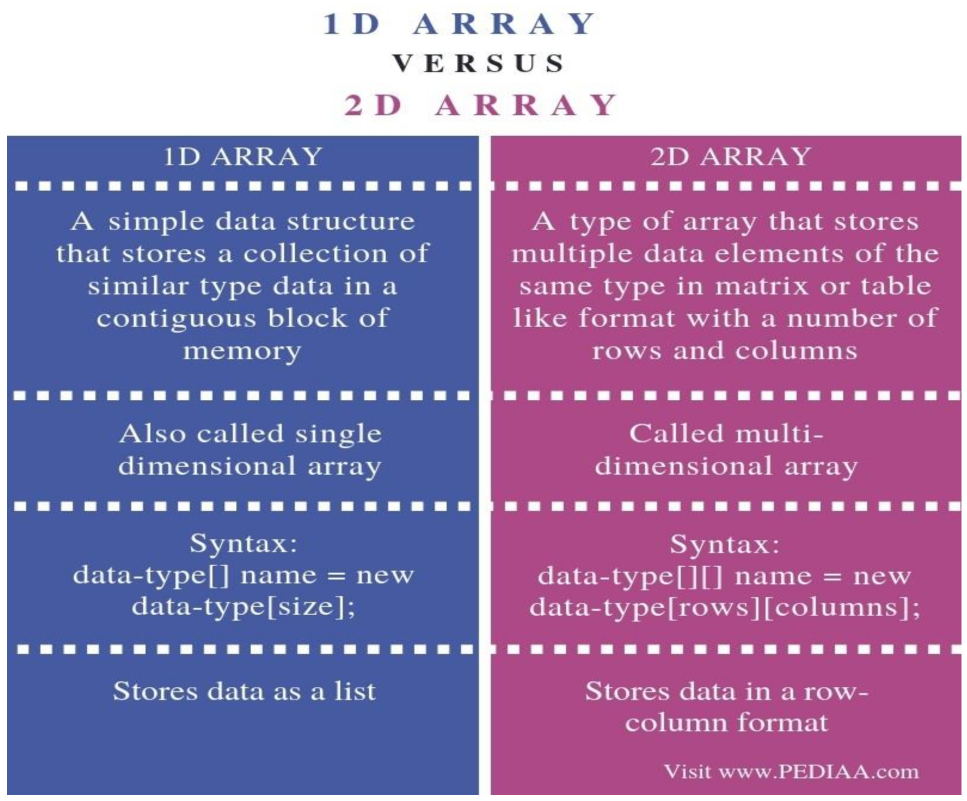
Here is how you can initialize two-dimensional and three-dimensional arrays:

Initialization of a 3D array

You can initialize a three-dimensional array in a similar way like a two-dimensional array. Here's an example,

```
int test[2][3][4] = {
 {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
 {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

Similarly, you can declare a three-dimensional (3d) array.
For example,
int a[2][4][3]; Here, the array y can hold 24 elements.



1D ARRAY VERSUS 2D ARRAY

1D ARRAY	2D ARRAY
A simple data structure that stores a collection of similar type data in a contiguous block of memory	A type of array that stores multiple data elements of the same type in matrix or table like format with a number of rows and columns
Also called single dimensional array	Called multi-dimensional array
Syntax: data-type[] name = new data-type[size];	Syntax: data-type[][] name = new data-type[rows][columns];
Stores data as a list	Stores data in a row-column format

Visit www.PEDIAA.com

// C program to find the sum of two matrices.

```
#include<stdio.h>
void main()
{
        int a[50][50],b[50][50],c[50][50],i,j,m,n;
        printf("How many rows:\n");
        scanf("%d”,&m); // rows
        printf("How many columns:\n");
        scanf("%d”,&n); // columns
        printf("\n Enter first matrix:\n");

        for(i=0;i<m;++i)
                for(j=0;j<n;++j)
                        scanf("%d",&a[i][j]); // reading a matrix
        printf("\nEnter second matrix:\n");

        for(i=0;i<m;++i)
                for(j=0;j<n;++j)
                        scanf("%d",&b[i][j]); // reading b matrix

        for(i=0;i<m;++i)
                for(j=0;j<n;++j)
                        c[i][j]=a[i][j]+b[i][j]; // sum of two matrix
printf("\nMatrix after addition:\n");
for(i=0;i<m;++i)
        {
                for(j=0;j<n;++j)
                {

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

                printf("\n");
        }

} //main close
```

Output:

```
How many rows
2
How many columns
2
Enter first matrix:
1
2
3
4
Enter second matrix:
1
2
3
4
Matrix after addition:
2       4
6       8
```

Matrix Multiplication in C

- Here is the program for matrix multiplication in C.
- m and n are rows and columns of first matrix.
- p and q are rows and columns of second matrix.
- Then, multiplication is possible only if n==p.

```
#include<stdio.h>

int main()
{
        int  a[50][50],b[50][50],c[50][50],m,n,p,q,i,j,k;
        printf("Enter rows and columns of first matrix:\n");
        scanf("%d%d",&m,&n);
        printf("Enter rows and columns of second matrix:\n");
        scanf("%d%d",&p,&q);

        if(n==p)
        {
                printf("\nEnter first matrix:\n");

                for(i=0;i<m;++i)
                        for(j=0;j<n;++j)
                                scanf("%d",&a[i][j]); // reading a matrix
        printf("\nEnter second matrix:\n");

                for(i=0;i<p;++i)
                        for(j=0;j<q;++j)
                                scanf("%d",&b[i][j]); // reading b matrix

                printf("\nThe matrix multiplication is:\n");

                for(i=0;i<m;++i)
                {
                        for(j=0;j<q;++j)
                        {
                                c[i][j]=0;
                                for(k=0;k<n;++k)
                                        c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
                                                printf("%d\t ",c[i][j]);
                        }

                        printf("\n");
                }
        }
        else
                printf("\nSorry!!!! Matrix multiplication can't be done");


} // main close
```

<u>Output:</u>

```
Enter rows and columns of first matrix:3
3
Enter rows and columns of second matrix:3
3

Enter first matrix:
1 2 3
4 5 6
7 8 9

Enter second matrix:
9 8 7
6 5 4
3 2 1

The new matrix is:
30      24      18
84      69      54
138     114     90
```

Transpose of Matrix in C

- Here is the program for transpose of matrix in C.
- We first read a matrix of size mxn and then find its transpose by just interchanging the rows and columns i.e. rows become columns and columns become rows.

//Transpose of Matrix in C

```
#include<stdio.h>
void main()
{
        int a[50][50],i,j,m,n;
        printf("How many rows?:\n");
        scanf("%d",&n);
        printf("How many columns?:\n");
        scanf("%d",&m);
        printf("\nEnter the matrix:\n");
        for(i=0;i<m;++i)
                for(j=0;j<n;++j)
                        scanf("%d",&a[i][j]);
        printf("\nTranspose of given matrix:\n");

        for(i=0;i<m;++i)
        {
                for(j=0;j<n;++j)
                        printf("%d\t",a[j][i]);
                printf("\n");
        }
 }
```

Output

```
How many rows?3
How many columns?3

Enter the matrix:
1 2 3
4 5 6
7 8 9

Transpose of given matrix:
1 4 7
2 5 8
3 6 9
```

STRING

It is a collection of characters.
(OR)

In C programming, a string is a sequence of characters terminated with a null character \0. For example:

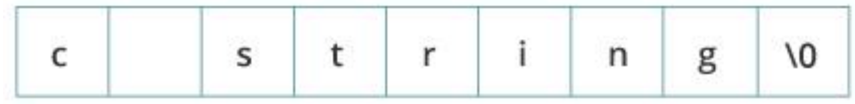
Syntax:

char string_name[size];

Example:

```
char  c[10] = "c string";
```

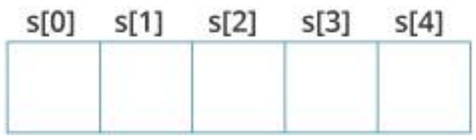
When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.



How to declare a string?

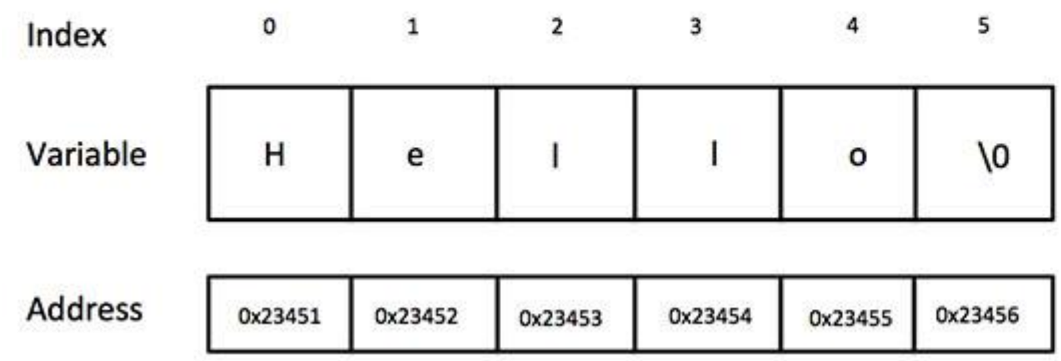
Here's how you can declare strings:

```
char s[5];
```



Here, we have declared a string of 5 characters.

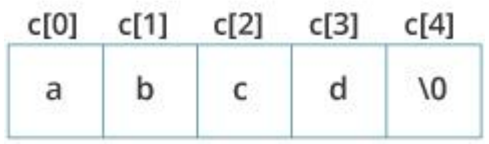
char str[6] = "Hello";



How to initialize strings?

You can initialize strings in a number of ways.

```
char c[10] = "abcd";
// string (by default null character added)


char c[5] = {'a', 'b', 'c', 'd', '\0'};
// character by charter
```



Let's take another example:

```
char c[5] = "abcde"; //error
```

Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters. This is bad and you should never do this.

Note: string size must be equal to length of string +1(NULL character);

Read String from the user

- You can use the scanf() function to read a string.
- The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab etc.).

Example 1: scanf() to read a string

```
#include <stdio.h>
void  main()
{
   char  name[20];
   printf("\n Enter name: ");
   scanf("%s", &name);
   printf("Your name is %s.", name);
   }
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Even though Dennis Ritchie was entered in the above program, only "Ritchie" was stored in the name string. It's because there was a space after Dennis.

How to read a line of text?

- You can use the gets() function to read a line of string with white space

  Syntax:
  ```
  char  ch[20];
  gets(ch);
  ```

- And, you can use puts() to display the string.

Sytax:

```
puts(ch);
```

Example 2: gets() and puts()

```
#include <stdio.h>
int  main()
{
   char  name[30];
   printf("Enter name:\n");
   gets(name)  // read string
   printf("\n string name: ");
   puts(name);    // display string
   return  0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

It's because gets() allows you to input any length of characters. Hence, there might be a buffer overflow.

String Manipulations In C Programming Using Library Functions

- Like integer's string can't copy, Concatenates and compare.
- The string manipulation can't be done automatically
- The string manipulation can done manually using string handling functions
- To solve this, C supports a large number of string handling functions in the standard library "string.h".

Few commonly used string handling functions are discussed below:

Strings handling functions are defined under "string.h" header file.

```
#include <string.h>
```

Note: You have to include the code below to run string handling functions.

All String Manipulation Function Given Below

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.

4	**strcmp(s1, s2);**
	Returns 0 if s1 and s2 are the same;
	less than 0 if s1<s2; (negative value)
	greater than 0 if s1>s2.(positive value)

5	**strrev(s1)** returns reverse string.
6	**strlwr(s1)** returns string characters in lowercase.
7	**strupr(s1)** returns string characters in uppercase**.**
8	**strchr(s1, ch);**

	Returns a pointer to the first occurrence of character ch in string s1.
9	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

Program: (FOR strcpy(), strcat() and strlen() functions)

```
#include <stdio.h>
#include <string.h>

void main ()
{
   char str1[50] = "Hello";
   char str2[50] = "World";
   char str3[50];
   int  len ;
   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );

   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );

   /* total lenghth of str1 after concatenation */
   len=strlen(str1);
   printf("strlen(str1) :  %d\n", len );
}
```

Output:

```
strcpy( str3, str1) :  Hello

strcat( str1, str2):   HelloWorld
```

```
strlen(str1) :  10
```

Program: (FOR strcmp() functions)

// Write c program to check two strings are equal or not

```
#include<stdio.h>
#include <string.h>
void main()
{
  char str1[20],str2[20];
  printf("Enter 1st string: ");
  gets(str1);//reads string from console
  printf("Enter 2nd string: ");
  gets(str2);
  if(strcmp(str1,str2)==0)
      printf("Strings are equal");
  else
      printf("Strings are not equal");
}
```

Output:

```
Enter 1st string: hello
Enter 2nd string: hello
Strings are equal
```

Program: (FOR strrev() functions)

// Write c program reverse the string

```
#include<stdio.h>
#include <string.h>
int main()
{
  char  str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nReverse String is: %s",strrev(str));
 return 0;
}
```

Output:

```
Enter string: javatpoint
String is: javatpoint
Reverse String is: tnioptavaj
```

Program: (FOR strlwr() functions)

// Write c program convert the string into lower case characters

```
#include<stdio.h>
#include <string.h>
int  main()
{
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nLower String is: %s",strlwr(str));
 return 0;
}
```

Output

```
Enter string: JAVATpoint
String is: JAVATpoint
Lower String is: javatpoint
```

Program: (FOR struper() functions)

// Write c program convert the string into upper case characters

```
#include<stdio.h>
#include <string.h>
int main()
{
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nUpper String is: %s",strupr(str));
 return 0;
}
```

Output:

```
Enter string: javatpoint
String is: javatpoint
Upper String is: JAVATPOINT
```

C String strstr()

The strstr() function returns pointer to the first occurrence of the matched string in the given string.

It is used to return substring from first match till the last character.

<u>Syntax:</u>

String strstr() parameters

string: It represents the full string from where substring will be searched.

match: It represents the substring to be searched in the full string.

<u>Program: (FOR String strstr() functions)</u>

<u>// Write a c program to print the string from sub string</u>

<u>Example</u>

```
#include<stdio.h>
#include <string.h>
int  main()
{
  char str[100]="this is javatpoint with c and java";
  char  *sub;
  sub=strstr(str,"java");
  printf("\nSubstring is: %s",sub);
 return 0;
}
```

<u>Output</u>

```
javatpoint with c and java
```

Program: (imp)

// Write a c program to reverse the string and find the given string is palindrome or not

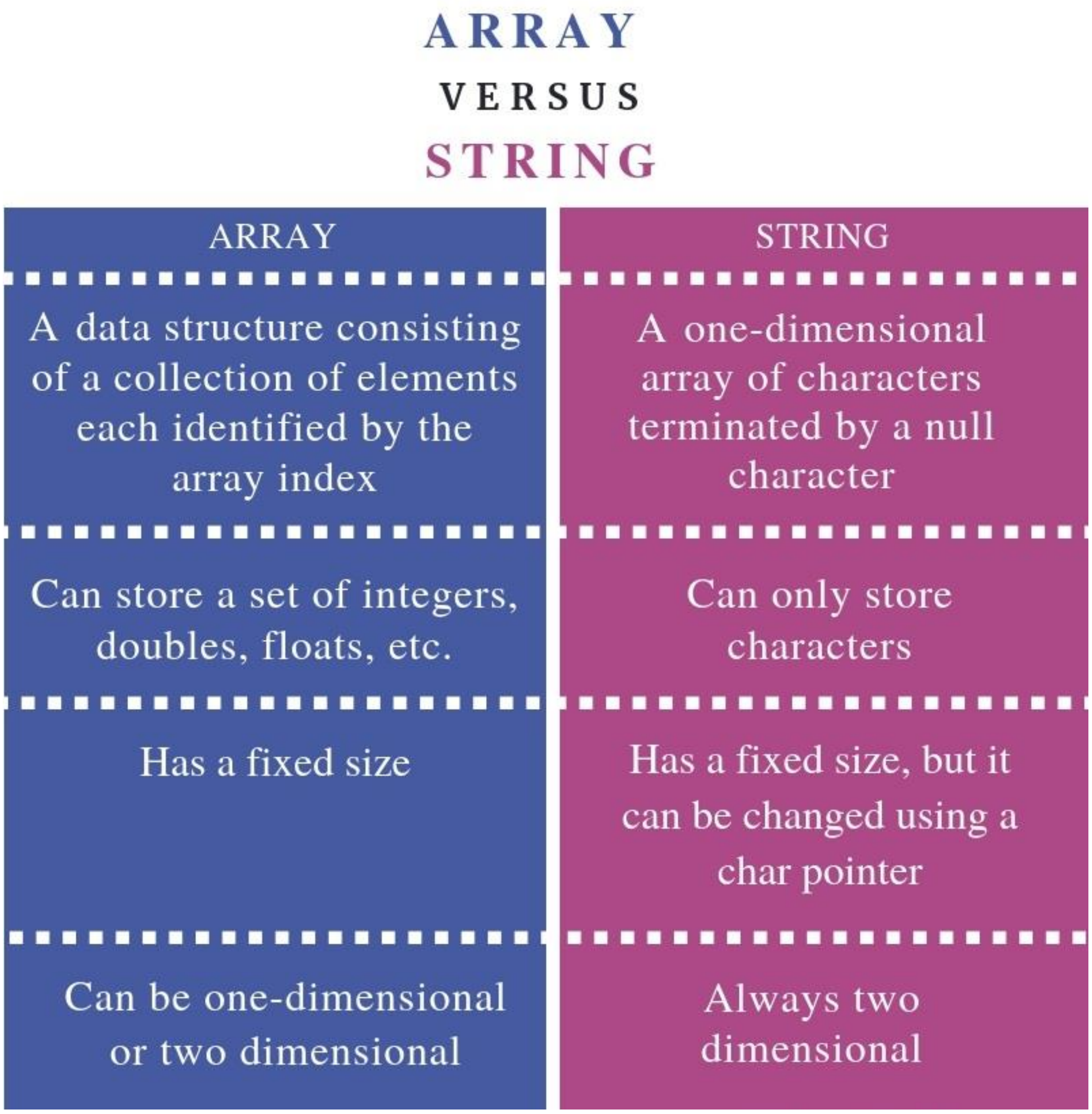
```
#include <stdio.h>
#include <string.h>
void main()
{
  char a[100], b[100];

  printf("Enter a string :\n");
  gets(a);

  strcpy(b, a);  // Copying input string
  strrev(b);  // Reversing the string

printf("\n reveres string is %s",b);

  if (strcmp(a, b) == 0)  // Comparing two strings
    printf("\n %s is string is a palindrome.\n",a);
  else
    printf("\n %s string isn't a palindrome.\n",a);

  }
```

Output:

```
Enter a string :
liril
```

reveres string is **liril**
liril is string is a palindrome.

<u>Difference between array and string</u>


ARRAY

VERSUS

STRING

ARRAY	STRING
A data structure consisting of a collection of elements each identified by the array index	A one-dimensional array of characters terminated by a null character
Can store a set of integers, doubles, floats, etc.	Can only store characters
Has a fixed size	Has a fixed size, but it can be changed using a char pointer
Can be one-dimensional or two dimensional	Always two dimensional

<u>Difference between array and string</u>

BASIS FOR COMPARISON	ARRAY	STRING
Basic	Character array is collection of variables, of character data type.	String is class and variables of string are the object of class "string".
Syntax	char array_name [size];	string string_name[size];
Indexing	An individual character in a character array can be accessed by its index in array.	In string the particular character can be accessed by the function "string_name.charAt(index)".
Data Type	A character array does not define a datatype.	A string defines a datatype in C++.

BASIS FOR COMPARISON	ARRAY	STRING
Operators	Operators in C++ can not be applied on character array.	You may apply standard C++ operator on the string.
Boundary	Array boundaries are easily overrun.	Boundaries will not overrun.
Access	Fast accessing.	Slow accessing.

Array of strings

A string is a 1-D array of characters, so an array of strings is a 2-D array of characters.

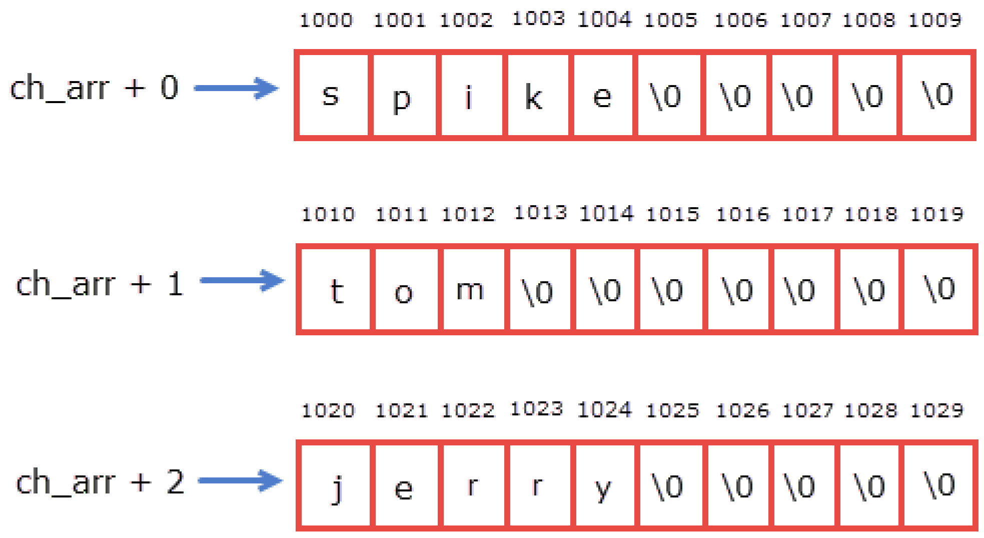
Declaration:

char variable_name[ROWS][COLS];

Here,
ROWS - Total number of maximum strings
COLS - Total number of characters in a string

char ch_arr[3][10] = {

```
                {'s', 'p', 'i', 'k', 'e'},
                {'t', 'o', 'm'},
                {'j', 'e', 'r', 'r', 'y'}
              };
```



/*C - Initialize Array of Strings in C,
C Program for of Array of Strings.*/

```
#include <stdio.h>

void main()
{

    char string[3][30]={"PPS","JAVA","PYTHON"};
    int i;

    //print all strings
    for(i=0;i<3;i++)
          {
      printf("%s\n",string[i]);
```

```
        }
}
```

Output

```
PPS
JAVA
PYTHON
```

String array using the array of pointer to string:

Pointer to array of string: A pointer which pointing to an array which content is string, is known as pointer to array of strings.

Syntax:

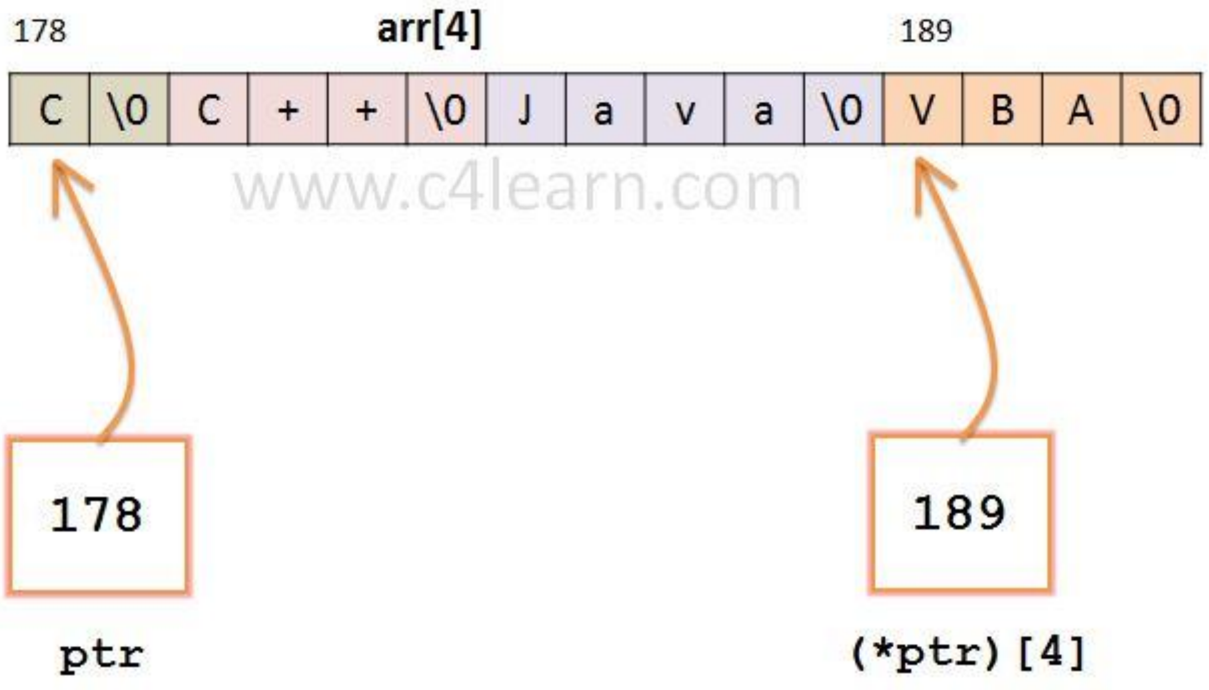
```
char  *arr[ROW]; //array of pointer to string
```

Example:

```
char  *arr[4] = {"C","C++","Java","VBA"};
```

In this example

ptr   : It is pointer to array of string of size 4.
array[4] : It is an array and its content are string.



Program: Printing Contents of character array

```
#include<stdio.h>
int  main()
{
int   i;

char  *arr[4] = {"C","C++","Java","VBA"};
char  *(*ptr)[4] = &arr;

for(i=0;i<4;i++)
   printf("String %d : %s\n",i+1,(*ptr)[i]);

return  0;
}
```

Output:

```
String 1 = C
String 2 = C++
```

String 3 = Java
String 4 = V

C Math (BULIT IN FUNCTIONS)

C Programming allows us to perform mathematical operations through the functions defined in <math.h> header file.
The <math.h> header file contains various methods for performing mathematical operations such as sqrt(), pow(), ceil(), floor() etc.

C Math Functions

There are various methods in math.h header file. The commonly used functions of math.h header file are given below.

No.	Function	Description
1)	ceil(number)	rounds up the given number. It returns the integer value which is greater than or equal to given number.
2)	floor(number)	rounds down the given number. It returns the integer value which is less than or equal to given number.
3)	sqrt(number)	returns the square root of given number.
4)	pow(base, exponent)	returns the power of given number.
5)	abs(number)	returns the absolute value of given number.

<u>Example program on C-Maths Function</u>

```
#include<stdio.h>
#include <math.h>
int main()
{
printf("\n%f",ceil(3.6));
printf("\n%f",ceil(3.3));
printf("\n%f",floor(3.6));
printf("\n%f",floor(3.2));
printf("\n%f",sqrt(16));
printf("\n%f",sqrt(7));
printf("\n%f",pow(2,4));
printf("\n%f",pow(3,3));
printf("\n%d",abs(-12));
 return 0;
}
```

<u>Output:</u>

```
4.000000
4.000000
3.000000
3.000000
4.000000
2.645751
16.000000
27.000000
```

12

STRUCTURES

We use structures to store data of different types.

- For example, you are a student. Your name is a string and your phone number and roll no are integers. So, here name, address and phone number is those different types of data. Here, structure comes in picture.

Defilations:

1. C Structure is a collection of different data types which are grouped into a single type (single name).
2. Structure is a user-defined data type in C language which allows us to combine data of different types together.

Syntax of structure

To define a structure, the **struct** keyword is used.

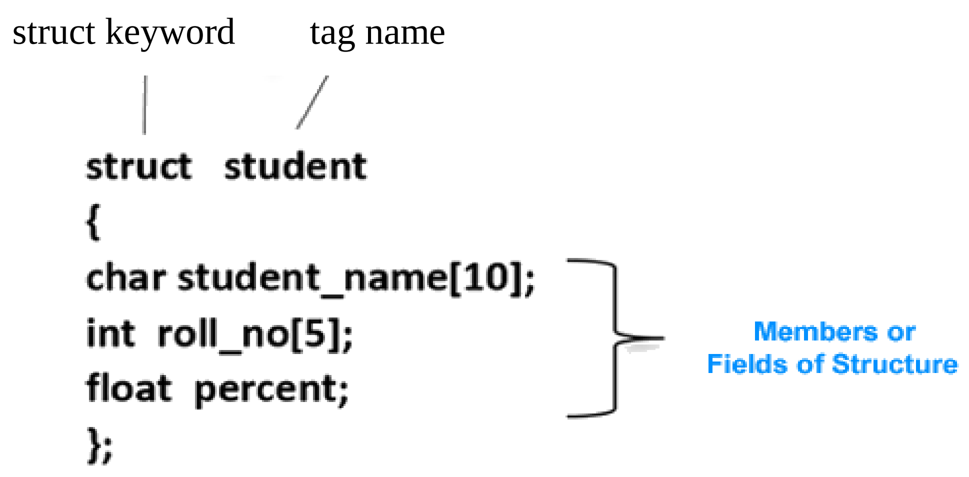
```
struct structureName

{

   dataType member1;

   dataType member2;     // Members of structure

   ...

dataType member n;

};
```

Where

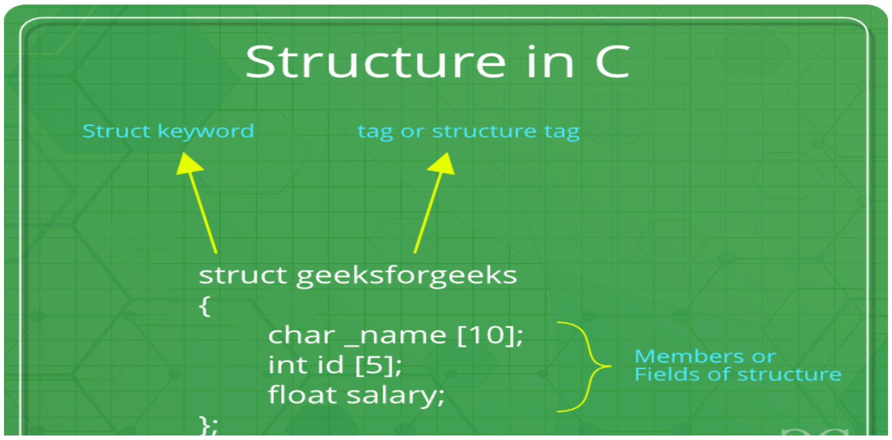
- **Struct** is key word
- **structureName** is user defined name ex: stud,emp,a,b
- dataType member1;
  dataType member2 are Members of structure

<u>Example:</u>

```
struct  employe

{

   char name[50];

   int eid;

   float salary;

};
```



```
struct   student
{
char student_name[10];
int  roll_no[5];
float  percent;
};
```



Structure in C

```
struct geeksforgeeks
{
        char _name [10];
        int id [5];
        float salary;
};
```

Defining a structure…

- The structure definition and variable declaration can be combined as:

```
struct student
        {
        char name[20];
        int roll_no;
        float marks;
        char gender;
        long int phone_no;
        }st1, st2, st3;
```

The use of *structure_name* is optional.

```
struct
{
char name[20];
int roll_no;
float marks;
char gender;
long int phone_no;
}st1, st2, st3;
```

<u>Structure Declaration has 3-ways:</u>

1. structure declaration with tag name
2. structure declaration without tag name
3. Declaring Structure using Typedef (user define)

1. structure declaration with tag name

```
struct employe

{

   char name[50];

   int eid;

   float salary;

};

struct employee e;
```

2. structure declaration without tag name

```
struct

{

   char name[50];

   int eid;

   float salary;

}e;
```

3. Declaring Structure using Typedef (user define)

```
typedef struct
{
  char  ename[30];
  int  ssn;
  int  deptno;
}employee;
```

OR

```
typedef  struct  Record
 {
    char ename[30];
    int ssn;
    int deptno;
 }employee;
```

```
employee e1 = {"raju",1000,90}; // Structure Variable
```

Declaration of Structure Variable (Create struct variables)

When a **struct** type is declared memory (storage) will not allocated for the members of structure. To allocate memory of a given structure type, we need to create structure variables.

Here's how we create structure variables:

1. **Using structure name**

```
struct employe
{
   char name[10];
   int eid;
   float salary;
};
 struct employe e1,e2,e3;
```

2. **Without Using structure name**

```
struct employe
{
   char name[50];
   int eid;
   float salary;
} e,e1,e2;
```

3. **In main function**

```
struct employe
{
   char name[10];
   int eid;
   float salary;
```

```
    };
int  main()
{
struct  employe e1,e2,e3; // structure variables.
   return  0;
}
```

Access members of a structure in main function:

There are two types of operators used for accessing members of a structure.

1. Using dot (.) operator (Member operator)

   Ex: e1.salary

2. Using selection(-> Structure pointer) operator

   ➔ - Structure pointer operator

   Example: e1-> name

Structure Initialization

1. Compile Time

This is wrong way of initialization

```
struct student
{
   char name[20]= “mahesh”;
   int  roll=67;
   float  marks=78.3;
}std1;
```

// this is wrong

Way 1 : Declare and Initialize

```
struct student
{
   char  name[20];
   int  roll;
   float  marks;
}std1 = { "hello",67,78.3};
```

Way 2 : Initializing inside main

```
struct student
{
   int mark1;
   int mark2;
   int mark3;
};

void main()
{
struct student s1 = {89,54,65};
- - - - --
- - - - --
- - - - --
};
```

// Runtime Initialization

- using scanf () funtion

Accessing members of a structure,Using dot (.) operator

(Member operator)

- Structure variable. Members of structure;

    Ex: e1.salary

Program: Store Information and Display it Using Structure(imp)

```
#include <stdio.h>
struct student
{
    char name[50];
    int rollno;
    float marks;
} s;
void main()
{
    printf("Enter the Student information:\n");
    printf("Enter name:\n ");
  scanf(“%s”,&s.name);
//gets(s.name);
    printf("\n Enter roll number: ");
    scanf("%d", &s.rollno);
    printf("\n Enter marks: ");
    scanf("%f", &s.marks);
    printf("Displaying student Information:\n");
    printf("\n Name:%s ",s.name);
    printf("\n Roll number: %d", s.rollno);
    printf("\n Marks: %f\n", s.marks);
}
```

OUTPUT:

```
Enter Student information:

Enter name: raju

Enter roll number: 23

Enter marks: 34.5

Displaying Student Information:

Name: raju

Roll number: 23

Marks: 34.5
```

Nested Structures:

- Nested structure in C is nothing but **structure within structure**.
- One structure contain the another structure variable as members the that structure called nested Structures

Syntax for structure within structure or nested structure

```
struct structure1
        {
            data type menber1;
            - - - - - - - - - -
        };

 struct structure2
        {
            datatype member2;
            - - - - - - - - - -
            struct structure1 obj1;
        }obj2;
```

<u>Accessing nested structure variable</u>

1. Outer structure variable. Inner structure variable. Inner Members of structure

Example:

obj2.obj1.member1;

<u>Example:</u>

```
struct address
{
   char  city[20];
   int  pin;
   char  phone[14];
};
struct  employee
{
   char  name[20];
   struct  address  add;
}emp;

emp.add.city; // nested structure
emp.name
```

// **<u>nested structure program</u>**

```
#include<stdio.h>
struct address
{
   char  city[20];
   int  pin;
};
struct  employee
{
   char  name[20];
   float  salary;
   struct  address add;
};
void main ()
{
   struct employee  emp;
   printf("\n Enter employee information?:\n");
  printf("\n Enter employee name and salary?:\n");
   scanf("%s %f”,&emp.name,&emp.salary);
printf("\n Enter employee city and pincode:\n");
   scanf(“%s%d”,&emp.add.city, &emp.add.pin);  // nested structure  members
   printf("\nPrinting the employee information....\n");
   printf("\n name: %s”, emp.name);
  printf("\n salary: %f”, emp.salary);
  printf("\nCity: %d\n pin:%d \n”, emp.add.city,emp.add.pin);
}
```

Output:

```
Enter employee information?
 Enter employee name and salary?:
Arun
25000
Enter employee city and pincode
hyderabad
500055
Printing the employee information....
name: Arun
salary: 25000.000000
city: hyderabad
Pincode: 500055
```

Array of Structures

Structure is collection of different data type.

An object of structure represents a single record in memory, if we want more than one record of structure type, we have to create an **array of structure or object.** As we know, an array is a collection of similar type; therefore an array can be of structure type.

Syntax:

```
struct struct_name
        {
            datatype var1;
            datatype var2;
            - - - - - - - - - -
            datatype varN;
        };
 struct struct_name obj [ size ];
```

Example:

```
struct car
{
   char make[20];
   char model[30];
   int year;
};
```

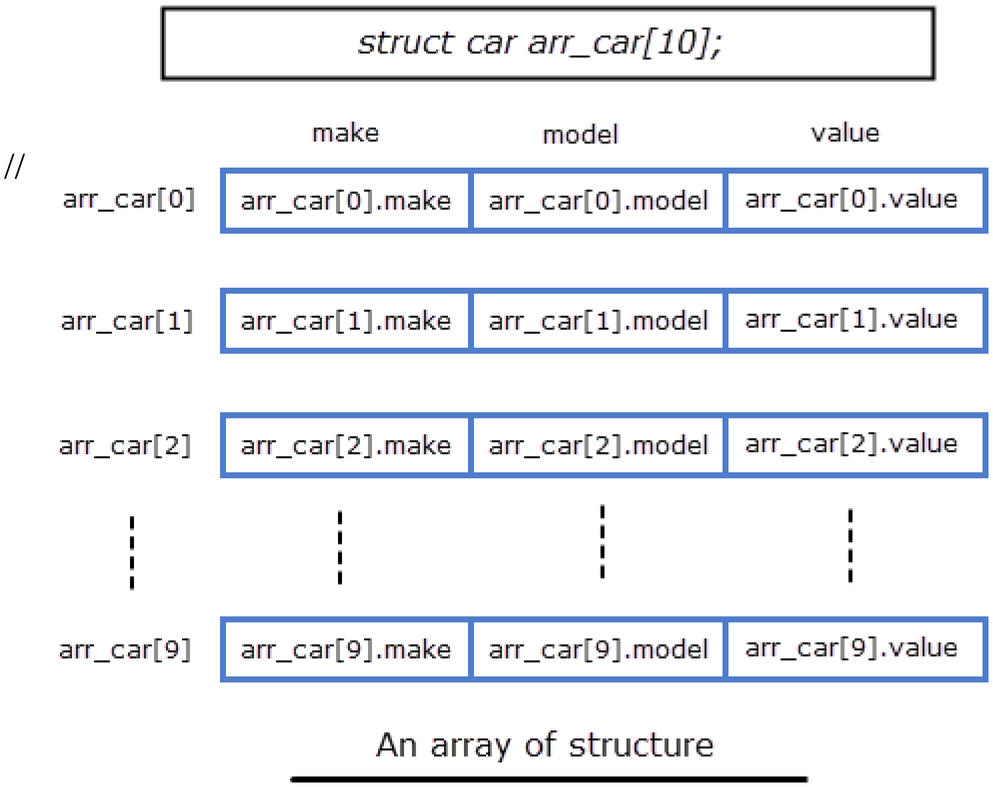
Here is how we can declare an array of structure car.

```
struct car c[10];
```

Accessing Array of Structures

```
c[i].make
c[i].made
```

//



An array of structure

Array of Structures program

```
#include<stdio.h>

struct student
{
   char name[20];
   int roll_no;
   float marks;
};
```

```
int  main()
{
   struct  student  stud[50];
   int  i,n;
printf (“\n How  many student records you want:\n”);
scanf(“%d”,&n);

   for(i = 0; i < n; i++ )
   {
      printf("\nEnter details of student %d:", i+1);
      printf("\n Enter name: ");
      scanf("%s", stud[i].name);
      printf("\n Enter roll no: ");
      scanf("%d", &stud[i].roll_no);

      printf("\n Enter marks: ");
      scanf("%f", & stud [i].marks);
   }

   printf("\nName:\tRoll no:\tMarks:\n");

   for(i = 0; i <n; i++ )
   {
 printf("%s\t%d\t%.2f\n",stud [i].name, stud [i].roll_no, stud [i].marks);
   }
return 0;
} // main close
```

Output:

```
How  many student records you want:

2

Enter details of student 1
Enter name: Jim
Enter roll no: 1
Enter marks: 44

Enter details of student 2
Enter name: Tim
Enter roll no: 2
Enter marks: 76
Name Roll no Marks
Jim    1        44.00
Tim    2        76.00
```

<u>Difference between Arrays and structure</u>

Arrays	Structures
1. An array is a collection of related data elements of the same type (homogenous).	1. Structure can have elements of different types (heterogeneous)
2. An array is a derived data type	2. A structure is a programmer-defined(user defined) data type
3. Any array behaves like a built-in data types. All we have to do is to declare an array variable and use it.	3. But in the case of structure, first, we have to design and declare a data structure before the variable of that type are declared and used.
4. Array allocates static memory and uses index/subscript for accessing elements of the array.	4. Structures allocate dynamic memory and uses (.) operator for accessing the member of a structure.
5. An array is a pointer to the first element of it	5. Structure is not a pointer
6. Element access takes relatively less time.	6. Property access takes relatively large time.

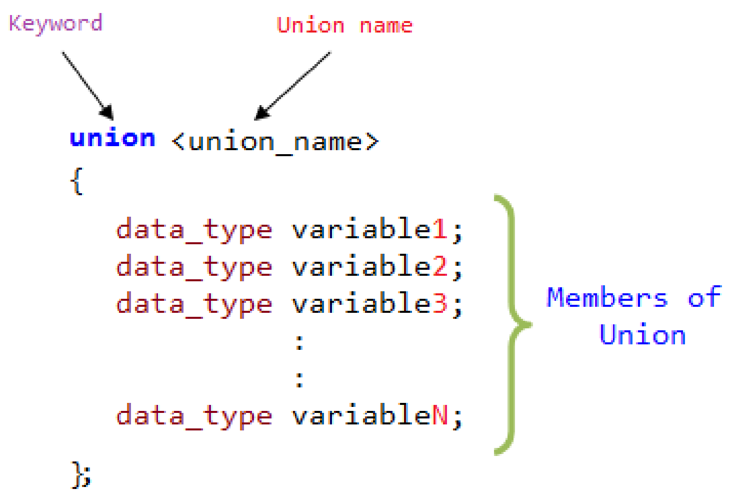
UNION

Unions are conceptually similar to structures.

Definition:

A union is a special data type available in C that allows storing different data types in the same memory location.

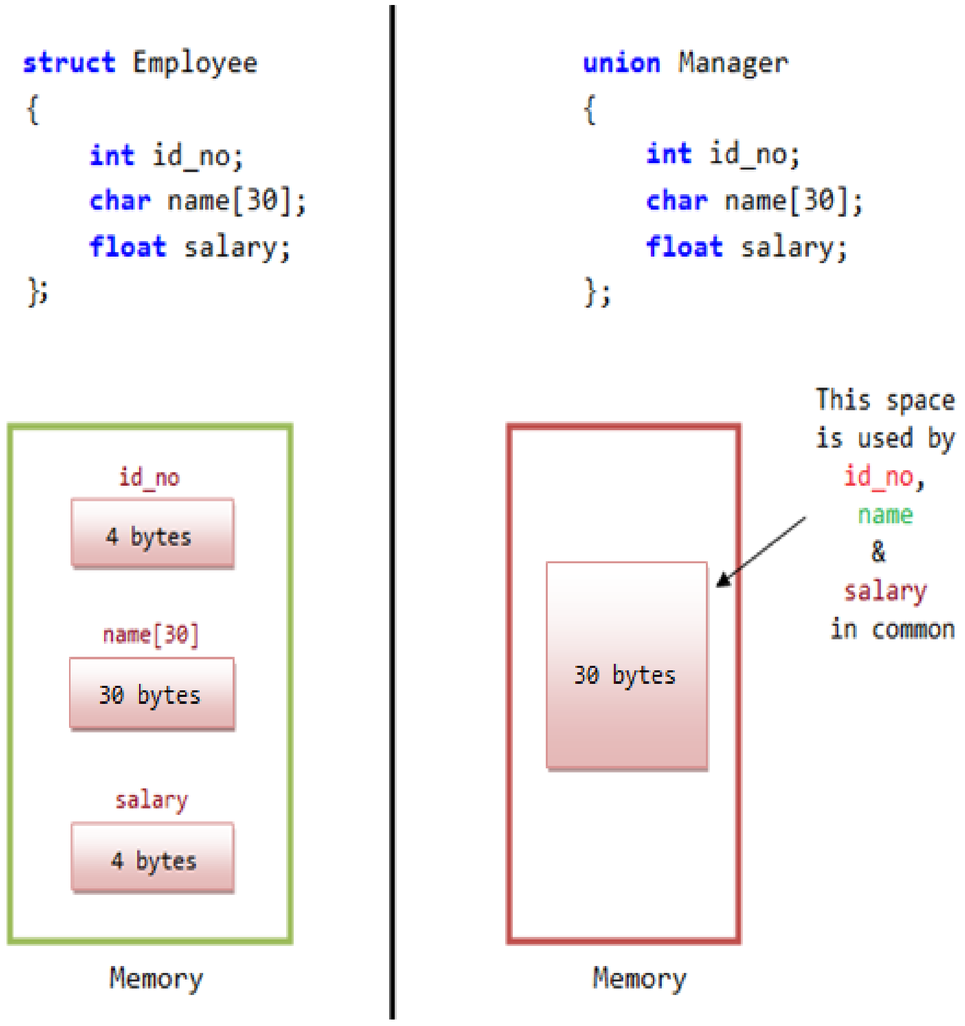
Syntax:



```
union <union_name>
{
   data_type variable1;
   data_type variable2;
   data_type variable3;
                :
                :
   data_type variableN;
};
```

Union union_name variable;

- In **structure** each member has its own storage location.
- Whereas all members of **union** use a single shared memory location, which is equal to the size of its largest data member.



```
struct Employee
{
    int id_no;
    char name[30];
    float salary;
};
```

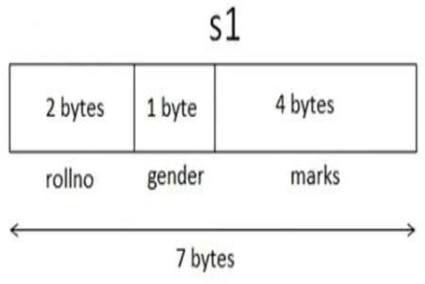
```
union Manager
{
    int id_no;
    char name[30];
    float salary;
};
```

• Union

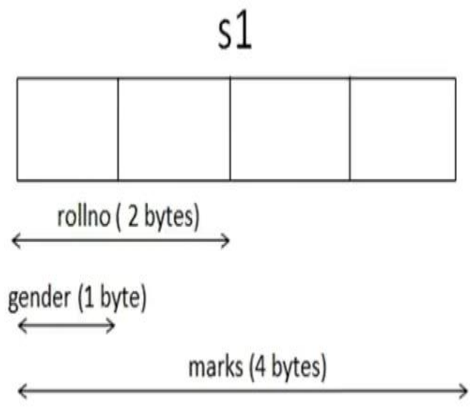
- Collections of dissimilar types of data.

- All members share same memory location.

```
union student {
        int rollno;
        char gender;
        float marks;
} s1;
```



Memory Allocation in Structure



Memory Allocation in Union

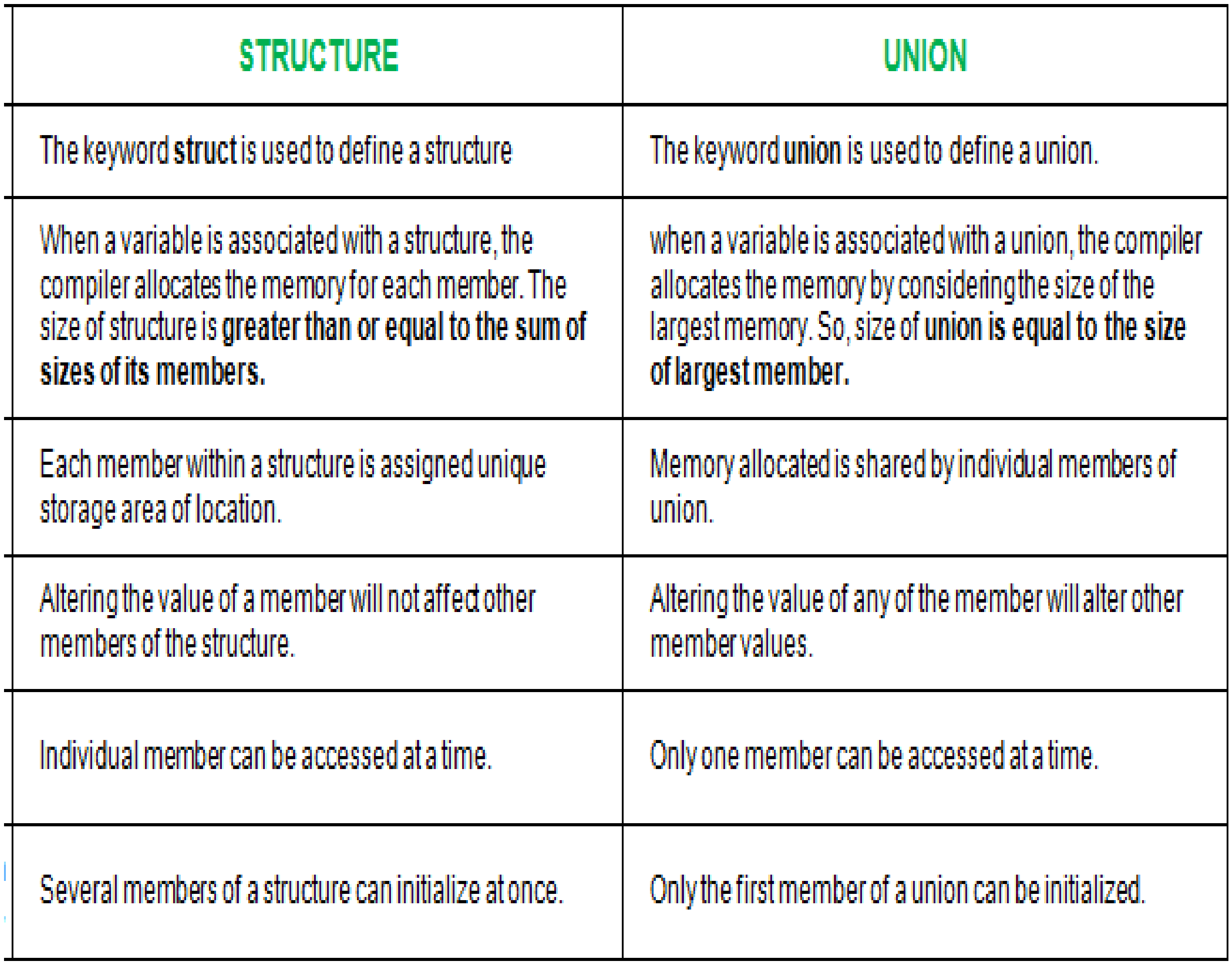
// Union example

```
#include <stdio.h>
union  Data
{
   int i;
   float f;
   char str[20];
};

int  main( )
{

   union  Data  data;

   printf( "Memory size occupied by data : %d\n", sizeof(data));

   return 0;
}
```

Output:

```
Memory size occupied by data : 20
```

Difference between structure and union



STRUCTURE	UNION
The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by consideringthe size of the largest memory. So, size of **union is equal to the size of largest member.**
Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Individual member can be accessed at a time.	Only one member can be accessed at a time.
Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Enumeration data type

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain. The keyword ‘enum’ is used to declare new enumeration types in C

An enum is defined in the same way as structure with the keyword struct replaced by the keyword enum and the elements separated by 'comma' as follows.

Syntax:

```
enum tagname {value1, value2, value3,....};
```

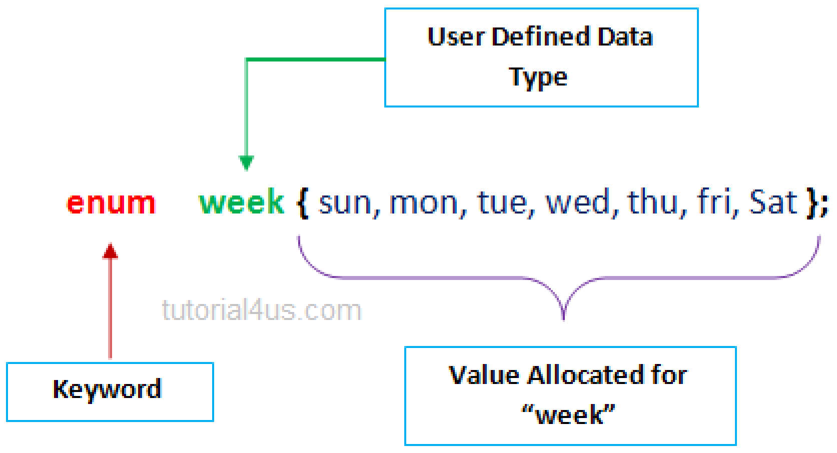
- In above syntax enum is a keyword. It is a user defiend data type.
- In above syntax tagname is our own variable. tagname is any variable name.
- value1, value2, value3,.... are create set of enum values.

Example:

```
enum week
{
 Sun,
mon,
 tue,
 wed,
 fri,
sat,

};
```

Or

```
enum week{ sun,mon,tue,wed,fri,sat};
```


User Defined Data Type
enum week { sun, mon, tue, wed, thu, fri, Sat };
tutorial4us.com
Keyword
Value Allocated for "week"

- ➢ It is start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.
- ➢ If constant one value is not initialized then by default sequence will be start from zero and next to generated value should be previous constant value one.

Example:

enum week {sun, mon, tue, wed, thu, fri, sat};

enum week today;

- ➢ today variable is declare as week type which can be initialize any data or value among 7 (sun, mon,....).

Example1:

```
#include<stdio.h>
```

```
enum week {sun, mon, tue, wed, thu, fri, sat};

void  main()
{
 enum  week today;
 today=tue;
 printf("%d day",today+1);

}
```

```
Output:
3 day
```

Example2:

```
#include<stdio.h>

enum week {sun, mon, tue, wed, thu, fri, sat};
void main()
{
for(i=sun; i<=sat; i++)
{
 printf("%d ",i);
}

}
```

```
Output:
1 2 3 4 5 6 7
```