Array

Array is a collection or grouping of elements with same datatype.

Example :-

(i) A college made of 'n' number of students with student numbers

(ii) An organisation made of 'n' number of employee with employee number.

Syntax to declare an array:-

<datatype> <array_name> [size];

The address of array is base address.

Advantage :-

Instead of declaring 'n' number of variables when all are of same datatype they can be grouped into a single array variable so that it can reduce the number of variables, time and length of the code

Initialisation of array:-

```
int a[5] = {2, 4, 6, 8, 10}
```

(i) The name of the array is itself the base or starting address of the array.

(ii) In array if the number of elements are 'n' then these elements will be stored from 0 to (n-1) location.

(iii) The array elements will be stored in ~~subs~~ sequential memory location.

Q. WAP to print the base address and address of all elements in the array.

```
# include <stdio.h>
# include <conio.h>
   void main()
{
int a[5] = {2, 3, 5, 7, 11};
 int i;
Printf ("The base address of a is:%d", &a);
   for (i=0; i<5; i++)
{
Printf ("\n The address of %d is %d", a[i], &a[i]);
}
  getch();
}
```

Q. WAP to accept a decimal numbers and convert it into its binary equivalent.

```
# include <stdio.h>
# include <conio.h>
  Void main ()
  {
  int rem, i = 1, n, bin = 0;
  Scanf ("%d", &n);
  while (n>0)
    {
    rem = n%2;
     n = n/2;
    bin = bin + (rem * i);
     i = i * 10;
    }
Printf ("%d", bin);
 getch ();
}
```

Accessing the array elements:-

```
  Void main ()
  {
  int a[5] = {3, 5, 7, 9, 11};
   int i;
for (i=0; i<5; i++)
   {
Printf ("%d", a[i]);
   }
getch ();
  }
```

Q. WAP to print the sum and average of 5 floating elements present in an array.

```
# include <stdio.h>
# include <conio.h>
  void main ()
   {
   int i;
  float a[5] ,sum=0, avg;
  for (i=0; i<5; i++)
   {
  scanf ("%f", &a[i]);
   }
  for (i=0; i<5; i++)
  {
  sum = sum+ a[i];
  avg = ave sum/5;
   }
  Printf ("%f %f", sum, avg);
   getch ();
  }
```

Q. WAP to print the minimum and maximum value from the given array elements.

```
# include <stdio.h>
# include <conio.h>
  void main()
   {
  int a[n], max, min;
   int i;
  for (i=0; i<n; i++)
```

```
        scanf ("%d", &a[i]);
        }
        min = a[0];
        max = a[0];
    for (i=1; i<n; i++)
        {
        if ( a[i] < min)
        {
    printf (" The minimum value is : %d", a[i]);
            min = a[i];
        }

        if ( a[i] > max)
        {
    printf (" The maximum value is : %d", a[i]);
        max = a[i];
        }

      getch ();
    }
```

Searching :-

1. **Linear search :-**
It is also called brute-force approach.
Here comparision is done 'n' times for 'n' no. of ~~variables~~ elements in an array.

2. **Binary search :-**
Here comparision is done $\log_2 n$ times for 'n' no. of elements in an array

Q. WAP to search an element from an array

```
# include <stdio.h>
# include <conio.h>
void main ()
{
int a[10], sc, i;
for (i=0; i<10; i++)
  {
  scanf ("%d", &a[i]);
  }
scanf ("%d", &sc);
for (i=0; i<10; i++)
  {
  if (sc == a[i])
Printf ("The number is found at : %d", i);
  }
getch ();
}
```

Sorting :-

1. Linear Sorting
2. Bubble Sorting

2-D Array :-

It is also called as Matrix.

Syntax :-

<datatype> <array name> [Rowsize] [Column size];

Int marks [2][3] = { {30, 40, 50} {70, 75, 80} };

Q. WAP to accept two matrices and display the addition in third matrix.

```
# include <stdio.h>
# include <conio.h>
 void main ()
 {
 int a[3][3], b[3][3], c[3][3], i,j;
 for (i=0; i<3; i++)
   {
 for (j=0 ; j<3; j++)
   {
 scanf ("%d", &a[i][j]);
   }
   }
 for (i=0 ; i<3 ; i++)
  {
 for (j=0 ; j<3 ; j++)
  {
 scanf ("%d", & b[i][j]);
   }
  }
 for (i=0; i<3 ; i++)
   {
 for (j=0; j<3 ; j++)
   {
 c[i][j] = a[i][j] + b[i][j];
   }
 }
 Printf (" The addition of two matrices is %d");
```

```
for (i=0; i<3; i++)
  {
  for (j=0; j<3; j++)
  {
Printf ("%3d", c[i][j]);
  }
Printf ("\n");
  }
  getch();
}
```

Strings:-

(i) String is a collection or grouping of the characters which is terminated by null character ('\0').
(ii) The String is having its own format specifier i.e %s.
(iii) The name of the string itself is its address as well as the string.

Declaration and Initialization:-

```
char str[10] = "GEEK";
```

Reading and the string from the keyboard:-

1. using scanf()

```
scanf("%s", str);
```

It is not required to pass the address (&).

2. using gets(str);

Printing the String:-

1. Using printf()
   printf ("%s", str);

2. Using puts();
   puts(str);

The <String.h> header file provides a set of built in function to manipulate the String data.
some built in functions are strlen(), strcpy(), strcmp(), strrev(), strcat().

(i) Strlen()
It is used to determine the length of string.
char Str[10] = "GEEK BWP"
Strlen = 8

(ii) Strcpy (str1, str2)
It is used to copy the String. str2 gets copied on str1.

(iii) Strcmp()
It is used to compare two strings. It checks length and each character ~~and~~. It does not ignores the case.

(iv) Strcmpi()
It is also used to compare two strings. It checks length and each character. It ignores the case of characters.

(v) Strrev()

It is used to reverse the string.

(vi) Strcat (str1, str2)

It is used to concatnet the two string. str2 gets concatte to str1.

Strlength:-

```
void main ()
 {
 char str [20] ;
 int length = 0 ;
 for (i=0 ; str[i] != '\0' ; i++)
    {
    length++ ;
    }
 Printf ("In The length of String %s is : %d", str, length)
 }
```

Compare two strings:-

```
void main ()
 {
 char str1 [15], str2[20] ;
 int i = 0 ;
 gets (str1) ;
 gets (str2) ;
while (str1[i] == str2[i] && str1[i] != '\0')
    {
    i++ ;
    }
```

```
if ( str1[i] - str2[i] )
printf ("The strings are not equal");
  else
printf ("The strings are equal");
  }
```

Function:-

Function is a self content block of statements to perform a particular task.

Advantages of using function:-

1. Modularity
2. Reusability

Types of function:-

1. Library functions
2. User defined functions.

The function should be defined by using the following steps according to ANSI:-

1. Prototype function declaration
2. Function calling.
3. Function definition.

```
void add ()
{
int a,b,c;
```

```
scanf ("%d %d", &a, &b);
c = a+b;
return c;
}
```

3. No return type with argument:-

```
void add (int, int);
Void main ()
{
int a, b;
Printf ("\n Enter a & b");
scanf ("%d %d", &a, &b);
add (a, b);
}
void add (int x, int y)
{
int z;
z = x+y;
Printf ("\n Sum is : %d", z);
}
```

4. With return type with argument:-

```
int add (int, int);
void main ()
{
int a, b, c;
Printf ("\n Enter a & b");
scanf ("%d %d", &a, &b);
c = add (a, b);
```

```
printf ("%d", c);

int add (int x, int y)
 {
 int z;
 z = x+y;
 return z;
 }
```

In
1. The function Called called parameters are called Actual parameters and definition parameters are called as formal parameters.

2. Whenever the function is called the actual parameters will be copied to function definition of formal parameters.

3. No two ∯ function name should be same (according to C-language)

4. Every ∯ function should return only one output value.

Q. WAP to find the factorial of a number using function

```
# include <stdio.h>
# include <conio.h>
 void fact ();
 void main ()
 {
 int a, b;
Printf ("\n Enter the value of a");
```

```
Void main ()
{
add ();
}

Void add ()
{
int a, b, c;
Printf ("\n Enter a & b");
Scanf ("%d%d", &a, &b);
   c = a+b;
Printf ("Sum is %d", c);
}
```

The function definition can be of four types:-

1. No return type no argument

2. With return type no argument

```
void add ();
Void main ()
{
int sum;
sum = add ();
Printf ("\n sum is: %d", sum);
}
  int add ()
  {
  int a, b, c;
Printf ("\n Enter the number a&b");
```

```
scanf ("%d", &a);
b = fact (a);
printf ("%d", b);

void fact (int x)
{
int c = 1, i;
for (i = 1; i <= x; i++)
{
c = c * i;
}
return c;
}
```

Q. WAP to define a function max() which returns the maximum of three integer values.

```
# include <stdio.h>
# include <conio.h>
int max (int, int, int)
void main()
{
int a, b, c, m;
printf ("\n Enter the value a, b & c");
scanf ("%d %d %d", &a, &b, &c);
m = max (a, b, c);
printf ("The maximum value among %d %d & %d is: %d",
a, b, c, m);
getch();
}
int max (int x, int y, int z)
```

```
{
return((x>y)&&(y>z)?x:(y>z)?y:z)
}
```

Function Recursion:-

(i) If the same function calls itself repeatedly until a certain condition is called as recursive function.

(ii) The recursive function should be stopped using if condition.

(iii) The recursive function is more efficient and reduces the length of the code.

Q. Write a recursive function to return the factorial value of a given number.

```
# include <stdio.h>
# include <conio.h>
void main ()
int n, f;
int fact ();
void main ()
{
int n, f;
Printf ("\n Enter the value of n");
Scanf ("%d", &n);
f = fact (n);
Printf ("\n The factorial is %d", f);
getch ();
}

int fact (n)
{
if (n==0 || n==1)
return (1);
```

```
else
return (m * fact(m-1));
}
```

Array as an argument in function:-

```
Void main ()
 {
 int A[20], Se, i, C;
Printf ("\n Enter the elements list");
 for (i=0 ; i<20 ; i++)
  {
  Scanf ("%d", &A[i]);
  }
Printf ("\n Enter the searching element");
  Scanf ("%d", &Se);
 C = B_search (A, Se, 20);
  if (C == -1)
Printf ("\n The element is not found in the
          list");
       else
Printf ("\n The element is found at position : %d",
          C);
  getch();
 }

int B_Search (int A[], int Se, int n)
 {
 int l=0, u=19, mid;
 while (l<=u)
  {
```

```
mid = (l+u)/2;
if (se == A[mid])
return mid;
else if (se < A[mid]),
    u = mid-1;
else
    l = mid+1;
}
return -1;
}
```

Passing of argument in function:-
It is of two types:-

1. Pass by valve / call by valve.
2. Pass by reference / call by reference.

Call by value:-
Passing the valve of variables through the function is called the call by valve.
In called by valve if we will do any modification to the function definition of formal parameter that will not affect to the actual parameter of function call.

Call by reference:-
In passing by address of variable through the function call is called call by reference.
In call by reference if we will do any modification to the function definition of formal parameter that will affect to the function call actual parameter.

Pointer - It is a variable which holds the address of another variable.
It is denoted as '*x'.

Q. Write a function to convert a string from upper case to lower case.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main ()
 {
 char str;
 printf ("\n Enter the string");
  gets (str);
 to small (str);
  getch ();
 }

 void to small (char s)
  {
  int i;
 for (i=0; s[i]!='\0' ; i++)
  {
 if (s[i] >=65 && s[i] <=90)
    s[i] = s[i]+32 ;
  else
  continue ;
  }
 puts (s);
  }
```

Scope of variable:-

Whenever the variable is declared the scope or life-time of the variable may be limited to a particular block (function) or entire program.

The C-language supports four types of storage class:-

1. Auto (local variable)
2. Extern (global variable)
3. static
4. Register.

Auto:-

1. The scope or life of auto or local variable is limited to a particular block (function)
2. These variables will be stored in RAM.
3. The default value of auto or local variable is garbage.

Example:-

```
void f1();
void main()
 {
 int a=30, b=40;
Printf ("%d %d", a, b);
  f1();
Printf ("%d %d", a, b);
  getch();
 }
void f1()
 {
```

```
int a = 70, b = 90;
Printf ("%d %d", a, b);
    }
```

Extern :-

1. The scope or life of extern or global variable is throughout the entire program.
2. These variables will be stored in RAM.
3. The default values of extern or global variables is zero.

Example :-

```
        void f1();
extern int a, b;
        void main()
        {
        a = a + 30;
        b = b + 50;
Printf ("%d %d", a, b);
        f1();
Printf ("%d %d", a, b);
        }
        void f1()
        {
        a = a + 20;
        b = b + 40;
Printf ("%d %d", a, b);
        }
```

Static variables :-

1. It must be declared using the static keyword.
2. It will be stored in seperate static memory.
3. The default value of static variables is zero.
4. The static variables are initialized only once in the lifetime of the program.
5. The static variable declaration may be in a block but the scope or life of a static variable is throughout the entire program.

Uses :-

1. It will be used to count the number of times a particular function has been called.
2. It will be used to generate a sequence in memory 1,2,3...

Example :-

```
void f1();
void main()
 {
 int i;
for(i=0; i<3; i++)
 f1();
 }
void f1()
 {
 static int x=0;
  x++;
printf("%d", x);
 }
```

OUTPUT:
1, 2, 3

Register :-

1. Register is nothing but a memory location.
2. C-language supports to store the data into register memory also.
3. Register is the keyboard to declare register variable.
4. Register memory are fastes in execution.

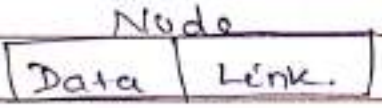
Example:

```
register int a = 10;
```

Linked List:-

```
Struct node
{
int data;
node *link;
};
```

Node



Structure:-

(i) Structure is a collection or grouping of variables with different datatype.

(ii) It is the organisation of data.

Example:-

(a) An employee structure consist of data like employee number, name, salary, designation etc.

(b) A student structure can hold the information about a student like student number, name, marks, total marks.

Average marks, grade etc.

(iii) The structure is an user defined data type.

(iv) 'Struct' is the keyword used to define the structure.

Syntax :-

```
Struct <name>
  {
  datatype 1 variable 1 ;
  datatype 2 variable 2 ;
    :
  };
```

Example :-

```
Struct emp
  {
  char emp_name [15] ;
  int emp_id;
  float sal ;
  char emp_desig [15];
  };
```

Structure declaration.

Struct emp e1, e2 ; → Structure variable declaration.

```
scanf ("%d", &e1.id);
gets (e1.emp-name);
```

Q. WAP to input two complex numbers calculate the sum of the two and store in another complex number.

```
Struct complex
{
float real, img;
};
complex complex_add (complex, complex);
void main ()
{
Struct complex c1, c2, c3;
Printf ("\n Enter the real and imaginary part of c1");
Scanf ("%f%f", &c1.real, &c1.img);
Printf ("\n Enter the real and imaginary part of c2");
Scanf ("%f%f", &c2.real, &c2.img);
    c3 = complex_add (c1, c2);
Printf ("\n %f + i%f", c3.real, c3.img);
  getch ();
}

complex complex_add (complex c1, complex c2)
  {
  complex c3;
  c3.real = c1.real + c2.real;
  c3.img = c1.img + c2.img;
    return c3;
  }
```

(i) Passing the structure variable through the function and returning a structure to a function.

(ii) Through the function we can pass any type of data and the type of data may be built in or user-defined (structure).

(iii) The function can return any output, the output can

also be built in or user defined type.

Array of structures :-
Array is a collection of elements of similar data-type similarly array of structures is the collection of elements with similar structure type.

Declaration :-

```
Struct emp
{
Char emp_name [15] ;
int id ;
float sal ;
char designation [15] ;
};
Struct emp e[100];
```

Q. WAP to consider a structure Student which consist of number, name and marks of three subject develope the program to accept the student details and check whether the student is pass or fail. If pass calculate the total and average and accordingly specify the grade of the student.

> 70	A
>60 && < 70	B
>=50 && < 60	C
< 50	D
< 30	F

```
Struct Student
{
Char name[15];
int num;
int marks[3];
int total;
float avg;
char grade;
};
void main()
{
Struct Student *S;
int n, i;
Printf("\n Enter the number of students");
Scanf("%d", &n);
S = (Student *) malloc (n * size of (student));
for (i=0; i<n; i++)
{
Printf("\n Enter the student name");
gets (S[i] → name);
Printf("\n Enter the number of students");
Scanf("%d", &S[i] → num);
Printf("\n Enter the marks of 3 subjects");
Scanf("%d %d %d", &S[i] → marks[0], &S[i] → marks[1],
&S[i] → marks[2]);
}
for (i=0; i<n; i++)
{
if (S[i] → marks[0] >= 90 && S[i] → marks[1] >= 90 &&
S[i] → marks[2] >= 90)
{
```

```
S[i]->total = S[i]->marks[0] + S[i]->marks[1] + S[i]->marks[2];
S[i]->avg = S[i]->total / 3;
if ( S[i]->avg >= 70 )
    S[i]->grade = 'A';
else if ( S[i]->avg >= 60 && S[i]->avg < 70)
    S[i]->grade = 'B';
else if ( S[i]->avg >= 50 && S[i]->avg < 60)
    S[i]->grade = 'C';
else ( S[i]->avg < 50)
    S[i]->grade = 'D';
}
else
    S[i]->grade = 'F';
}
```

Nested Structure :-

If any structure consist of variable which is the type of another structure than it is called as nested structure.

one structure inside another structure

Example :-

```
struct emp
{
char name[15];
char desg[10];
int id;
float sal;
date dob, doj;
};
struct date
```

```
{
int dd, mm, yy;
};
struct emp e;
```

Accessing the structure members:-

e.name, e.desg, e.id, e.sal, e.dob.dd, e.dob.mm, e.dob.yy, e.doj.dd, e.doj.mm, e.doj.yy

Union :-

(i) It is the grouping of variables like a Structure. It is also an user defined datatype.

(ii) Union is a keyword.

(iii) The difference between Structure and union is Structure allocates the memory for all the members where as union takes the maximum size of data member. That single memory will be used for all other members of the union so that we can save the memory.

Where Union is used - The union will be used whenever we require any one of the member at a time in different functions.

Selection sort → In Selection sort select the minimum or maximum value according to either ascending or descending order. Interchange the value and place the min or max value at the top of the array. This situation happens until you get the elements either in ascending or descending order.

```
Void Selection_sort (int *a, int n)
 {
int i, min, ~~maxx~~ minp, t;
for (i=0; i<n; i++)
{
minp = minimum (a,n,i)
 if (i != minp)
 {
 t = *a[i];
*a[i]=*a[minp];
*a[minp] = t;
 }
 }
}

int minimum (int *a, int n, int i)
 {
 int i, min, minp;
 min=*a[i];
 minp = i;
for (i= i+1 ; i<n ; i++ )
  {
 if (*a[i] < min)
   min = *a[i];
   minp = i;
  }
 return minp;
```

Running time of quick sort and merge sort is $n \log n$

Pointer

1. Pointer is a variable which holds the address of another variable.
2. Pointer is a variable which holds the address of memory.

Syntax :- datatype * variable.name;

```
int *aa;
float *ff;
char *cc;
void *ptr;
```

The format specifier for pointer variable is %u.

Operators in Pointer:-

1. Address operator :-
   It gives or ~~to~~ returns the address of a variable.

2. Pointer Operator:
It gives the value from address.

1. Any pointer variable occupies two bypes of memory even void pointer also because pointer is an unsigned integer type.

2. The integer pointer can hold the address of integer variable, float pointer can hold the address of float variable etc.

3. But the void pointer can holds the address of any one variable either integer or float or character variable etc thats why it is also called as generic pointer.

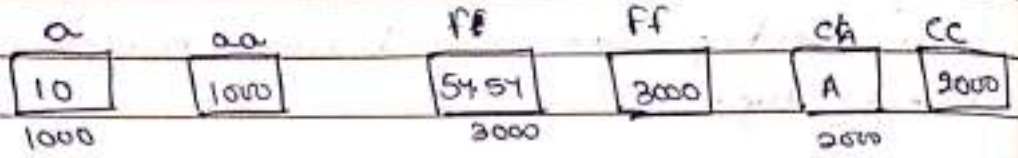
int a=10;

a → location name
| 10 | → value
&a → address

int *aa; (declaration)
aa=&a; (initialisation)

```
Void main ()
{
int a=10, *aa;
char ch='A', *cc;
float f=54.54, *ff;
aa = &a;
cc = &ch;
ff = &f;
Printf ("\n %d is stored at %u address", *aa, aa);
Printf ("\n %c is stored at %u address", *cc, cc);
```

```
printf ("in %f is stored at %u address", *ff, ff);
    getch();
}
```



Output:-

10 is stored at 1000 address
A is stored at 2000 address
54.54 is stored at 3000 address.

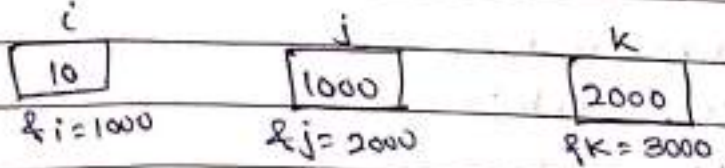
Pointer to pointer variable:-
It is a pointer variable which holds the address of another pointer variable.

Syntax:-

datatype **variable name;

Example:- int **a;

```
int i=o, *j, **k;
  j= &i;
  k= &j;
```



Accessing the address of i:-

&i = j = *k ;

Accessing the address of j:-

&j = k ;

Accessing the value of i:-

i = *j = **k ;

~~Dynamic declaration~~

Dynamic Memory Allocation:-

int a[25];

(i) This declaration tells the compiler that to create an integer type of memory with 25 elements i.e. 50 bytes into the base address of array (&a).

(ii) This memory will be allocated by the compiler at the compile time.

(iii) The compile time memory allocation is called as static memory allocation.

(iv) It is a fixed memory.

(v) It will raise two problems:-

(a) Whenever the user require less number of elements according to the declaration than the memory will be wasted (underflow of memory).

(b) If the user require more number of elements than

the memory will be overflow (shortage of memory)

To solve the above problem in C is offering dynamic memory allocation (DMA). This memory will be allocated at runtime and there will not be wastage or shortage of memory.

malloc() / calloc() are used to allocate memory dynamically.
These functions are declared and defined in <alloc.h> header file.

Malloc():-
Syntax :- <pointer variable> = (typecast *) malloc (no. of elements * size of (datatype));

```
int *a = (int *) malloc (15 * size of (int));
```

Calloc():-
Syntax :- <pointer variable> = (typecast *) calloc (no. of elements, size of (datatype));

```
int *a = (int *) calloc (15, size of (int));
```

Q. WAP to calculate the sum and average of n integers using dynamic memory allocation.

```
void main()
{
int *a, n, i, sum=0, avg;
Printf ("In Enter the number of elements");
```

```
Scanf("%d", &n);
a = (int *) malloc (n * Size of (int));
for (i=0 ; i<n ; i++)
Scanf ("%d", a[i]);
for (i=0 ; i<n ; i++)
  {
  Sum = Sum + *a[i] ;
  }
  avg = Sum/n ;
Printf("\n Sum is = %d", Sum);
Printf ("\n Average is = %d", avg);
  getch ();
}
```