ADDITION OF AN ARRAY ELEMENT USING SINGLE DIMENSIONAL ARRAY :

```
#include<stdio.h>
#include <conio.h>
void main ( )
{
    int i, a[5],sum=0;
    clrscr( );
    printf (“Enter the elements\n”);
    for (i=0; i< 5; i++)
   {
          scanf (“%d”, &a [i]);
          sum = sum + a[i];
}
printf (“The sum of the array elements is %d “,sum);
getch();
}
```

Input :

Enter the array elements

1

2

3

4

5

Output : The sum of the array elements is 15

/* PROGRAM TO REVERSE A NUMBER*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int rem,sum=0,n;
   clrscr();
   printf("enter number:\n");
   scanf("%d",&n);
   while(n>0)
  {
      rem=n%10;
      sum=(sum*10)+rem;
      n=n/10;
```

```
}
  printf("\n reversed number is:%d", sum);
  getch();
}
```

OUTPUT:

~~~~~~

```
enter number:123
reversed number:321
```

/*matrix Additon*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int m,n,p,q,i,j,a[10][10],b[10][10],c[10][10];
```
~~~~~~

```
clrscr();
printf ("ENTER THE ORDER OF THE MATRIX\n");
scanf ("%d  %d",&m,&n);
printf("ENTER THE VALUES FOR I MATRIX");
for(i=0; i<m; i++)
for(j=0; j<n; j++)
{
   scanf ("%d", &a[i][j]);
}
printf ("ENTER THE VALUES FOR II MATRIX");
for(i=0; i<m; i++)
for(j=0; j<n; j++)
{
   scanf ("%d", &b[i][j]);
 }

for ( i =0; i<m; i++)
for(j=0; j< n; j++)
{
   c[i][j] = a[i][j] + b[i][j];
}
```

```
   printf ("The output of added two matrix is \n");
   for (i=0; i < m; i++)
   for(j=0; j <n; j++)
    {
       printf ("%d\t\t\n", c[i][j]);
       }
       getch();
 }
```

NESTING OF FUNCTION :

A nested function is a function defined inside another function. The nested function's name is local to the block where it is defined.

PROGRAM EXAMPLE

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
   void  fun1( );  // declaration
   printf ("I am in main");
   fun1 ( );
```

```
    printf ("Again I am in main");
  }
void    fun1 (  )
{
     void   fun2(  );
     printf (" I am in funciton1");
     fun2(  );
}
void   fun2 (  )
{
    printf ("I am in function2");
}
```

Output :

```
I am in main
I am in function1
I am in function 2
Again I am in main
```

PASSING ARRAYS TO FUNCTIONS :

To pass an entire array to a function, only the name of the array is passed an an argument.

PROGRAM EXAMPLE

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
   int i, m, a[100];
    int big (int [ ] , int);
   printf ("Enter how many number");
   scanf("%d", &n);
   printf ("Enter numbers 1 by 1");
   for ( i = 0; i < n; i++)
   scanf ("%d", & a[ i ] );
   m = big (a , n);
   printf ("The maximum number is an array is %d", m);
}
int  big (int  x[100] ,  int  y)
{
    int  j, max;
    max  =  x [ 0 ];
    for ( j = 1;  j  <  y;  j ++ )
      if  (  max  <   x[ j ]  )
            max  =  x[  j  ];
             return (max);
   }
```

CATEGORIES OF FUNCTIONS :

NO ARGUMENTS NO RETURN VALUE :

```
/* FACTORIAL NUMBER*/
#include <stdio.h>
#include < conio.h>
void main ( )
{
   void   fact (  );
   clrscr (  );
   fact (  );
  getch( );
}
void   fact (  )
{
   int  f = 1, i, n;
  printf (“Enter the Number”);
 scanf (“%d”, &n);
 for ( i=1;  i < = n;   i++)
 {
      f   =   f  * i ;
}
printf (“ %d”, f);
```

```
}
```

WITH ARGUMENTS WITH RETURN VALUE

```
/* FACTORIAL NUMBER*/
#include <stdio.h>
#include < conio.h>
void main ( )
{
    int c , n;
    int  fact (int );
    clrscr (  );
    printf (“Enter the Number”);
    scanf (“%d”, &n);
    c  =  fact (  n  );
    printf (“%d”, c);
    getch( );
}

int   fact  (int n )
{
       int f= 1, i;
       for ( i = 1;   i < = n;  i++)
      {
```

```
            f  =  f  *  i;
        }
        return  ( f );
}
```

WITH ARGUMENTS NO RETURN VALUE

```
/* FACTORIAL NUMBER*/
#include <stdio.h>
#include < conio.h>
void main ( )
{
    int   n;
    void   fact (int );
    clrscr (  );
    printf (“Enter the Number”);
    scanf (“%d”, &n);
    fact (  n  );
    printf (“%d”, c);
    getch( );
}
void   fact  (int   n )
{
        int  f= 1, i;
        for ( i = 1;   i < = n;  i++)
```

```
    {
        f = f * i;
    }
    printf ("%d " , f );
}
```

RECURSIVE FUNCTION:

If a function calls itself within the function itself, it is called recursive function call.

Advantages :

- **Easy to understand.**
- **Easy and it is very compact to write.**
- **The recursive data structure is useful like a tree structure.**

Disadvantages :

- **It executes slowly.**
- **Additional storage space is required.**
- **Stack process is needed to process the recursion.**

PROGRAM EXAMPLE :

/* Factorial Number using Recursion*/

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

```
#include<conio.h>
void  main ( )
{
       Int x, f, fact (int);
       clrscr ( );
       printf (“Enter a Number”);
       scanf (“%d”, &x);
      f  =  fact (x);
     printf (“Factorial of   %d  is  %d”, x , y) ;
}
fact ( int m )
{
       int f,m;
     if ( m  == 0 )
          return (1);
   else
       f = m * fact (m-1);
       return ( f );
}
```

OUTPUT :

Enter a Number : 5

Factorial of 5 is 120.

PASSING STRINGS TO FUNCTIONS :

Rules : The string to be passed must be declared as a formal argument of the function when it is defined. Example :

```
void display (char it_name [ ])
{
      -----
}
```

For the above function definition, the prototype can be written as

```
void display (char str [  ]);
```

Example :

```
#include <stdio.h>
#include <conio.h>
void displaystr(char [ ]);
void main ( )
{
      char message[ ] = "WELCOME";
      displaystr(message);
}
```

```
void displaystr(char str[ ])
{
        printf ("String : %d\n", str);
 }
```

Output :

String : WELCOME

STORAGE CLASS SPECIFIERS :

It can be used to provides information about the variables location and visibility. Storage class specifiers can be used to declare explicitly the scope and lifetime of variables.

- **Automatic variables**
- **External variables**
- **Static variables**
- **Register variables**

Automatic variables : Automatic variables are declared inside a function in which they are to be utilized. They are created when the function is called and destroyed automatically when the function is exited, hence the name automatic. Local variable known only to the function in which it is declared. Default is auto.

Example :

```
main ( )
{
        auto int number;
          ------
}
```

External variables :

Variables that are both alive and active throughout the entire program are known as external variables. They are also known as global variables.

```
int number;
float length = 10.5;
main ( )
{
    -----
    -----
}
function1( )
{
   ----
```

```
        -----
    }
    funcion2( )
    {
            -----
            ------
    }
```

Static Variables :

Local variable which exists and retains its value after the control is transferred to the calling function. A static variable may be either an internal or an external type.

```
static int x;
static float y;
```

Register Variables :

Local variable which is stored in the register.

Ex : `register int count;`

Text Book :

Programming in ANSI C – 4E- E.BALAGURUSAMY