Head Recursion

```c
#include <stdio.h>

void fun(int n)
{
    if(n>0)
    {
        fun(n-1);
        printf("%d ",n);

    }
}

int main() {
    int x=3;

    fun(x);
    return 0;
}
```

Tail Recursion

```c
#include <stdio.h>

void fun(int n)
{
    if(n>0)
    {
       printf("%d ",n);
       fun(n-1);

    }
}

int main() {
    int x=3;
```

```
    fun(x);
    return 0;
}
```

Static Variables in Recursion

```c
#include <stdio.h>

int fun(int n)
{
    static int x=0;
    if(n>0)
    {
        x++;
        return fun(n-1)+x;
    }
    return 0;
}

int main() {

    int r;
    r=fun(5);
    printf("%d\n",r);

    r=fun(5);
    printf("%d\n",r);

    return 0;
}
```

Global Variabels in Recursion

```c
#include <stdio.h>


int x=0;

int fun(int n)
{
   if(n>0)
    {
```

```
        x++;
        return fun(n-1)+x;
    }
    return 0;
}

int main() {

    int r;
    r=fun(5);
    printf("%d\n",r);

    r=fun(5);
    printf("%d\n",r);

    return 0;
}
```

Tree Recursion

```c
#include <stdio.h>
void fun(int n)
{
    if(n>0)
    {
        printf("%d ",n);
        fun(n-1);
        fun(n-1);
    }
}
int main() {
    fun(3);
    return 0;
}
```

Indirect Recursion

```c
#include <stdio.h>

void funB(int n);

void funA(int n)
{
    if(n>0)
    {
        printf("%d ",n);
        funB(n-1);
    }
}
void funB(int n)
{
    if(n>1)
    {
        printf("%d ",n);
        funA(n/2);
    }
}

int main()
{
    funA(20);
    return 0;
}
```

Nested Recursion

```c
#include <stdio.h>

int fun(int n)
{
    if(n>100)
        return n-10;
    return fun(fun(n+11));
}

int main()
{
    int r;
    r=fun(95);
    printf("%d\n",r);
    return 0;
}
```

Sum of N natural numbers

```c
int sum(int n)
{
    if(n==0)
        return 0;
    return sum(n-1)+n;
}

int Isum(int n)
{
    int s=0,i;
        for(i=1;i<=n;i++)
                s=s+i;

    return s;

}
int main()
{
    int r=sum(5);
    printf("%d ",r);

    return 0;
}
```

Factorial of N

```c
int fact(int n)
{
    if(n==0)
        return 1;
    return fact(n-1)*n;
}

int Ifact(int n)
{
    int f=1,i;
        for(i=1;i<=n;i++)
                f=f*i;

    return f;
}
int main()
{
    int r=Ifact(5);
    printf("%d ",r);

    return 0;
}
```

Power Function

```
int power(int m,int n)
{
    if(n==0)
        return 1;
    return power(m,n-1)*m;
}

int power1(int m,int n)
{
    if(n==0)
        return 1;
    if(n%2==0)
        return                                  power1(m*m,n/
2);
    return m * power1(m*m,(n-1)/2);
}
int main()
{
    int r=power1(9,3);
    printf("%d ",r);

    return 0;
}
```

Taylor Series using Static variables

```c
double e(int x, int n)
{
    static double p=1,f=1;
    double r;

    if(n==0)
        return 1;
    r=e(x,n-1);
    p=p*x;
    f=f*n;
    return r+p/f;
}
int main()
{
    printf("%lf \n",e(4,15));
    return 0;
}
```

Taylor Series

```c
double e(int x, int n)
{
    static double p=1,f=1;
    double r;

    if(n==0)
        return 1;
    r=e(x,n-1);
    p=p*x;
    f=f*n;
    return r+p/f;
}
int main()
{
    printf("%lf \n",e(4,15));
    return 0;
}
```

Taylor Series Horner's Rule

```c
double e(int x, int n)
{
    static double s;
    if(n==0)
        return s;
    s=1+x*s/n;
    return e(x,n-1);

}
int main()
{
    printf("%lf \n",e(2,10));
    return 0;
}
```

Taylor Serie Iterative

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

```c
double e(int x, int n)
{
    double s=1;
    int i;
    double num=1;
    double den=1;

    for(i=1;i<=n;i++)
    {
        num*=x;
        den*=i;
        s+=num/den;
    }
    return s;
}
int main()
{
    printf("%lf \n",e(1,10));
    return 0;
}
```

Taylor Serie Iterative

```c
#include <stdio.h>

double e(int x, int n)
{
    double s=1;
    int i;
    double num=1;
    double den=1;

    for(i=1;i<=n;i++)
    {
        num*=x;
        den*=i;
        s+=num/den;
    }
    return s;
}
int main()
{
    printf("%lf \n",e(1,10));
    return 0;
}
```

Fibonacci

```c
#include <stdio.h>

int fib(int n)
{
    int t0=0,t1=1,s=0,i;

    if(n<=1) return n;

    for(i=2;i<=n;i++)
    {
        s=t0+t1;
        t0=t1;
        t1=s;
    }

    return s;
}

int rfib(int n)
{
    if(n<=1)return n;
    return rfib(n-2)+rfib(n-1);
}

int F[10];

int mfib(int n)
{
    if(n<=1)
    {
        F[n]=n;
        return n;
    }
    else
    {
        if(F[n-2]==-1)
            F[n-2]=mfib(n-2);
        if(F[n-1]==-1)
```

```
            F[n-1]=mfib(n-1);
        F[n]=F[n-2]+F[n-1];
        return F[n-2]+F[n-1];
    }
}

int main()
{
    int i;
    for(i=0;i<10;i++)
        F[i]=-1;

    printf("%d \n",mfib(5));
    return 0;
}
```

Combination Formula

```c
#include <stdio.h>

int fact(int n)
{
    if(n==0)return 1;
    return fact(n-1)*n;
}

int nCr(int n,int r)
{
    int num,den;

    num=fact(n);
    den=fact(r)*fact(n-r);

    return num/den;
}

int NCR(int n,int r)
{
    if(n==r || r==0)
        return 1;
    return NCR(n-1,r-1)+NCR(n-1,r);

}

int main()
{
    printf("%d \n",NCR(5,3));
    return 0;
}
```

Tower of Hanoi

```c
#include <stdio.h>

void TOH(int n,int A,int B,int C)
{
    if(n>0)
    {
        TOH(n-1,A,C,B);
        printf("(%d,%d)\n",A,C);
        TOH(n-1,B,A,C);
    }
}
int main()
{
    TOH(4,1,2,3);
    return 0;
}
```

Static vs Dynamic Arrays

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

int main()
{
    int A[5]={2,4,6,8,10};
    int *p;
    int i;

    p=(int *)malloc(5*sizeof(int));
    p[0]=3;
    p[1]=5;
    p[2]=7;
    p[3]=9;
    p[4]=11;

    for(i=0;i<5;i++)
        printf("%d ",A[i]);

    printf("\n");
    for(i=0;i<5;i++)
        printf("%d ",p[i]);

    return 0;
}
```

Array Size

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

int main()
{
    int *p,*q;
    int i;
    p=(int *)malloc(5*sizeof(int));
    p[0]=3;p[1]=5;p[2]=7;p[3]=9;p[4]=11;

    q=(int *)malloc(10*sizeof(int));

    for(i=0;i<5;i++)
        q[i]=p[i];

    free(p);
    p=q;
    q=NULL;

    for(i=0;i<5;i++)
        printf("%d \n",p[i]);

    return 0;
}
```

2D Array

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

int main()
{
    int A[3][4]={{1,2,3,4},{2,4,6,8},{1,3,5,7}};

    int *B[3];
    int **C;
    int i,j;


    B[0]=(int *)malloc(4*sizeof(int));
    B[1]=(int *)malloc(4*sizeof(int));
    B[2]=(int *)malloc(4*sizeof(int));


    C=(int **)malloc(3*sizeof(int *));
    C[0]=(int *)malloc(4*sizeof(int));
    C[1]=(int *)malloc(4*sizeof(int));
    C[2]=(int *)malloc(4*sizeof(int));

    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
            printf("%d ",C[i][j]);
        printf("\n");
    }

    return 0;
}
```