**Recursion**

**Definition: ‘’ It is when a function all itself to make the problem smaller. ‘’**

/\* To get the sum of the first 'n' number using recursion.  \*/

int sum(int n){

    if(n==0){

        return 0;

    }

    int prevsum = sum(n-1);

    return n + prevsum;

}

int main(){

    int n;

    cin>>n;

    cout<<sum(n);

    return 0;

}

/\* Calculate n raised to power of p. \*/

int power(int n, int p){

    if(p==0){

        return 1;

    }

    int PrePower = power(n, p-1);

    return n\*PrePower;

}

int main(){

    int n, p;

    cin>>n>>p;

    cout<<power(n, p);

    return 0;

}

/\*  find factorial of a number of n \*/

int fact(int n){

    if(n==0){

        return 1;

    }

    //int prevfact = fact(n-1);

    //return n \* prevfact;

    return n \* fact(n-1);

}

int main()

{

    int n;

    cin>>n;

    cout<<fact(n);

    return 0;

}

/\*  print the n th Fibonacci number \*/

int fib(int n){

    if(n==0 || n==1){   //base case

        return n;

    }

    return fib(n-1) + fib(n-2);

}

int main()

{

    int n;

    cin>>n;

    cout<<"Fibonacci-"<<n<<" is "<<fib(n)<<endl;

    return 0;

}

/\*  Check if an array is strictly increasing order or not.  \*/

bool sorted(int arr[], int n){

    if (n==1){

        return true;

    }

    bool restArray = sorted(arr+1, n-1);

    return (arr[0]<<arr[1] && restArray);

}

int main()

{

    int arr[]= {1,2,3,4,5};

    cout<<sorted(arr, 5)<<endl;

    return 0;

}

/\*  Print number till 'n' in Decreasing order. \*/

void dec(int n){

    if(n==0){

        return;

    }

    cout<<n<<endl;

    dec(n-1);

}

int main()

{

    int n;

    cin>>n;

    dec(n);

    return 0;

}

/\* Print number till '1 to n' in Increasing as well as Decreasing order \*/

void dec(int n){

    if(n==1){

        cout<<"1"<<endl;

        return;

    }

    cout<<n<<" ";

    dec(n-1);

}

void inc(int n){

    if(n==1){

        cout<<"1"<<" ";

        return;

    }

    inc(n-1);

    cout<<n<<" ";

}

int main(){

    int n;

    cin>>n;

    dec(n);

    inc(n);

    return 0;

}

/\* Find the First and last occurrence of an element using recursion. \*/

int firstocc(int arr[], int n, int i, int key){

    if(i==n){

        return -1;

    }

    if(arr[i]==key){

        return i;

    }

    return firstocc(arr,n,i+1,key);

}

int lastocc(int arr[], int n, int i, int key){

    if(i==n){

        return -1;

    }

    int restArray = lastocc(arr,n,i+1,key);

    if(restArray!=-1){

        return restArray;

    }

    if(arr[i]==key){

        return i;

    }

    return -1;

}

int main(){

    int arr[]={4,2,1,2,5,2,7};

    cout<<"first array : "<<firstocc(arr,7,0,2)<<endl;

    cout<<"last array : "<<lastocc(arr,7,0,2)<<endl;

    return 0;

}

**16.3 Recursion 7-Best Problem**

/\* Reverse a string using recursion " binod "  \*/

void reverse(string s){

    if(s.length()==0){      //base case

        return;

    }

    string ros = s.substr(1);

    reverse(ros);

    cout<<s[0];

}

int main(){

    reverse("binod");

    return 0;

}

/\*  "pippppiiiipi"  \*/

void replacePi(string s){

    if (s.length() == 0){

        return; // base case

    }

    if (s[0] == 'p' && s[1] == 'i'){

        cout << "3.14";

        replacePi(s.substr(2));

    }

    else{

        cout << s[0];

        replacePi(s.substr(1));

    }

}

int main(){

    replacePi("pippppiiiipi");

    return 0;

}

/\* Tower\_of\_Hanoi \*/

void towerofHanoi(int n, char scr, char dest, char helper){

    if(n==0){

        return; //base case

    }

    towerofHanoi(n-1,scr,helper,dest);

    cout<<"move from "<<scr<<" to "<<dest<<endl;

    towerofHanoi(n-1,helper,dest,scr);

}

int main(){

    towerofHanoi(3,'A','C','B');

    return 0;

}

/\* given string remove dublicate: "aaaabbbeeecdddd" \*/

string removeDup(string s){

    if(s.length()==0){

        return "";

    }

    char ch=s[0];

    string ans = removeDup(s.substr(1));

    if(ch==ans[0]){

        return ans;

    }

    return (ch+ans);

}

int main(){

    cout<<removeDup("aaaabbbeeecdddd");

    return 0;

}

/\*  Move all x to the end of the string " axxbdxcefxhix " \*/

string movellx(string s){

    if(s.length()==0){

        return "";

    }

    char ch=s[0];

    string ans = movellx(s.substr(1));

    if(ch=='x'){

        return ans+ch;

    }

    return ch+ans;

}

int main(){

    cout<<movellx("axxbdxcefxhix")<<endl;

    return 0;

}

/\*  Generate all substrings of a string " ABC " \*/

void subseq(string s, string ans){

    if(s.length()==0){

        cout<<ans<<endl;

        return;

    }

    char ch = s[0];

    string ros =s.substr(1);

    subseq(ros, ans);

    subseq(ros, ans+ch);

}

int main(){

    subseq("ABC", "");

    cout<<endl;

    return 0;

}

/\*  General substring with ASCII number str="AB",ASCII=65,66    \*/

void subseq(string s, string ans){

    if(s.length()==0){

        cout<<ans<<endl;

        return;

    }

    char ch = s[0];

    int code = ch;

    string ros =s.substr(1);

    subseq(ros, ans);

    subseq(ros, ans + ch);

    subseq(ros, ans + to\_string(code));

}

int main(){

    subseq("AB", "");

    return 0;

}

/\*  Print all possible word from phone digits   \*/

string keypadArr[] = {"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz","./"};

void keypad(string s, string ans){

    if(s.length()==0){

        cout<<ans<<endl;

        return;

    }

    char ch = s[0];

    string code = keypadArr[ch-'0'];

    string ros =s.substr(1);

    for(int i=0; i<code.length(); i++){

        keypad(ros, ans + code[i]);

    }

}

int main(){

    keypad("23","");

    return 0;

}

**16.4 Recursion Advanced Problems:**

Print all possible permutation of a string "ABC".

void permutation(string s, string ans){

    if(s.length()==0){

        cout<<ans<<endl;

        return;

    }

    for(int i=0; i<s.length(); i++){

        char ch = s[i];

        string ros = s.substr(0,i) + s.substr(i+1);

        permutation(ros, ans+ch);

    }

}

int main(){

    permutation("ABC", "");

    return 0;

}

Count the number of paths possible from start point to end point in gameboard.

int countpath(int s, int e){

    if(s==e){

        return 1;

    }

    if(s>e){

        return 0;

    }

    int count = 0;

    for(int i=1; i<=6; i++){

        count += countpath(s+i,e);

    }

    return count;

}

int main(){

    cout<<countpath(0,3);

    return 0;

}

Count the number of paths possible in a Maze.

int countpathMaze(int n, int i, int j){

    if(i==n-1 && j==n-1){

        return 1;

    }

    if(i>=n || j>=n){

        return 0;

    }

    return countpathMaze(n,i+1,j) + countpathMaze(n,i,j+1);

}

int main(){

    cout<<countpathMaze(3,0,0)<<endl;

    return 0;

}

**16.5 Recursion Last Lecture on Advanced Problems:**

Given a "2 X n" board and tiles of size "2 X 1", count the no. of the given board using these tiles.

int tilingWays(int n){

    if(n==0){

        return 0;

    }

    if(n==1){

        return 1;

    }

    return tilingWays(n-1) + tilingWays(n-2);

}

int main(){

    cout<<tilingWays(4);

    return 0;

}

**Output**: 3

Find the no. of ways in which n friends can remain single or can be paired up.

int friendsPairing(int n){

    if(n==0 || n==1 || n==2){

        return n;

    }

    return friendsPairing(n-1) + friendsPairing(n-2)\*(n-1);

}

int main(){

    cout<<friendsPairing(4);

    return 0;

}

Output: 10

Put n items with weight and value and value in a knapsack of capacity W to get the maximum total value in the knapsack.

int knapsack(int value[], int wt[], int n, int w){

    if(n==0 || w==0){

        return 0;

    }

    if(wt[n-1]>w){

        return knapsack(value, wt, n-1, w);

    }

    return max(knapsack(value, wt, n-1, w-wt[n-1]) + value[n-1],

                    knapsack(value, wt, n-1, w));

}

int main(){

    int wt[] = {10,20,30};

    int value[] = {100,50,150};

    int w = 50;

    cout<<knapsack(value,wt,3,w)<<endl;

    return 0;

}

Output: 250