|  |  |
| --- | --- |
| **Name** | Shubhan Singh |
| **UID no.** | 2022300118 |
| **Experiment No.** | 4 |

|  |  |
| --- | --- |
| **AIM:** | Apply the concept of recursion to solve a given problem. |
| **Program 1** | |
| **PROBLEM STATEMENT :** | Write a recursive function to find the factorial of a number and test it. |
| **ALGORITHM:** | Algorithm for main()  Step 1: START  Step 2: Read an integer n from input.  Step 3: fac=factorial(n)  Step 4: print the value of fac.  Step 5: END  Algorithm for factorial(int n)  Step 1: if n>1, return n\*factorial(n-1)  Step 2: if n equals 1 or 0, return 1 |
| **PROGRAM:** | #include<stdio.h>  long long factorial(int *n*){      if(*n*==0){return 1;}      if(*n*==1){return 1;}      return *n*\*factorial(*n*-1);  }  int main(){      int n;      long long fac;      printf("enter a non negative number\n");      scanf("*%d*",&n);      if(n<0){printf("invalid input\n");}      fac=factorial(n);      printf("The factorial of *%d* is *%lld*\n",n,fac);      return 0;  } |
| **RESULT:** | |
| **Program 2** | |
| **PROBLEM STATEMENT :** | Write a recursive function which returns the nth term of the fibonacci series. Call it from main() to find the 1st n numbers of the fibonacci series. |
| **ALGORITHM:** | Algorithm for main()  Step 1: START  Step 2: Read a number n from input  Step 3: Initialize i=1  Step 4: Print value of fib(i)  Step 5: Increment i  Step 6: If i<=n, return to step 4  Step 6: END  Algorithm for function fib(int n)  Step 1: if n>2, return fib(n-1)+fib(n-2)  Step 2: if n equals 2, return 1  Step 3: if n equals 1, return 0 |
| **PROGRAM:** | #include<stdio.h>  int fib(int *n*){      if(*n*==1){return 0;}      if(*n*==2){return 1;}      return fib(*n*-1)+fib(*n*-2);  }  int main(){      int n;      printf("Enter a number\n");      scanf("*%d*",&n);      printf("The first *%d* terms of Fibonacci series are as follows:\n",n);      for(int i=1;i<=n;i++){          printf("*%d*  ",fib(i));          if(i>1){          if(i%10==0){printf("\n");}          }          }          printf("\n");      return 0;  } |
| **RESULT:** | |
| **Program 3** | |
| **PROBLEM STATEMENT:** | Given a number n, print following a pattern without using any loop.  Example:  Input: n = 16  Output: 16, 11, 6, 1, -4, 1, 6, 11, 16  Input: n = 10  Output: 10, 5, 0, 5, 10 |
| **ALGORITHM:** | Algorithm for main()  Step 1: START  Step 2: Read a number n.  Step 3: execute patternrev(n)  Step 4: execute pattern(n)  Step 5: END  Algorithm for function patternrev(int n)  Step 1: print n  Step 2: if n>5, execute pattern(n-5)  Algorithm for function pattern(int n)  Step 1: if n>0, execute pattern(n-5)  Step 2: print n |
| **PROGRAM:** | #include<stdio.h>  void pattern(int *n*){      if(*n*>0){pattern(*n*-5);}      printf("*%d*  ",*n*);    }  void patternrev(int *n*){      printf("*%d*  ",*n*);      if(*n*>5){patternrev(*n*-5);}  }  int main(){      int n;      printf("Enter a number\n");      scanf("*%d*",&n);      patternrev(n);      pattern(n);        return 0;  } |
| **RESULT:** | |
| **Program 4** | |
| **PROBLEM STATEMENT:** | Ackerman’s function is defined by:  A(m,n) =n+1 if m=0  =A(m-1,1) if m≠0 and n=0  =A (  m-1   ,   A(m,n-1)    ) if m≠0 and n≠0  Write a function which given m and n returns A(m,n). |
| **ALGORITHM:** | Algorithm for main()  Step 1: START  Step 2: Read value of m and n from input.  Step 3: x=ackerman(m,n)  Step 4: Print value of x.  Step 5: END  Algorithm for int ackerman(int a, int b)  Step 1: if a equals 0, return b+1  Step 2: if a does not equal 0 and b equals 0, return ackerman(a-1,1)  Step 3: if both a and b do not equal 0, return ackerman(a-1,ackerman(a,b-1)) |
| **PROGRAM:** | //Program for returning A(m,n) as per given input  #include<stdio.h>  int ackerman(int *a*,int *b*){      int x;      if(*a*==0){x=*b*+1;}      if((*a*!=0) && *b*==0){x=ackerman(*a*-1,1);}      if((*a*!=0) && (*b*!=0)){x=ackerman(*a*-1,ackerman(*a*,*b*-1));}      return x;  }  int main(){      int x,m,n;      printf("Enter value of m and n respectively\n");      scanf("*%d* *%d*",&m,&n);      x=ackerman(m,n);      printf("The value of Ackerman function for *%d* and *%d* A(*%d*,*%d*) is: *%d*\n",m,n,m,n,x);      return 0;  }  //Program for tabular output  #include<stdio.h>  int ackerman(int *a*,int *b*){      int x;      if(*a*==0){x=*b*+1;}      if((*a*!=0) && *b*==0){x=ackerman(*a*-1,1);}      if((*a*!=0) && (*b*!=0)){x=ackerman(*a*-1,ackerman(*a*,*b*-1));}      return x;  }  int main(){      int x;      printf("Ackerman function table:\n\n");  */\*for formatting\*/*printf("       m=1         m=2         m=3\n\n");      for(int j=1;j<=10;j++){  */\*for formatting\*/*if(j<10){printf("n=*%d*    ",j);}      else{printf("n=*%d*   ",j);}          for(int i=1;i<=3;i++){              x=ackerman(i,j);              printf("A(*%d*,*%d*)=*%d*  ",i,j,x);  */\*This part is just for formatting\*/*              if(i==1){                  if(j<=7){printf("  ");}                  else if(j>7 && j<10){printf(" ");}                  }              else if(i==2){                  if(j<=3){printf("  ");}                  else if(j>3 && j<10){printf(" ");}                  }*/\*till here\*/*          }          printf("\n");      }      return 0;  } |
| **RESULT:** For basic program    For tabulated output | |
| **CONCLUSION:** | We studied the method of recursion for solving certain problems, and that it can act as a substitute to control structures even in problems that do not necessarily need it. |