|  |  |
| --- | --- |
| **Name** | Shubhan Singh |
| **UID no.** | 2022300118 |
| **Experiment No.** | 3 |

|  |  |
| --- | --- |
| **AIM:** | To apply the concept of functions to incorporate modularity |
| **Program 1** | |
| **PROBLEM STATEMENT :** | Write a function which takes a range as input. Print all the numbers in the range with ‘\*’ in front of prime numbers only. |
| **ALGORITHM:** | (The algorithm and flowchart have been written/made without the steps needed for the formatting of console output, as it was not mentioned in the question)  Step 1: START  Step 2: Read values for range and store them in A and B respectively.  Step 3: Set A=1 if A<1.  Step 4: Execute function primes(int a, int b) with A and B as arguments.  Step 5: END  Algorithm for function primes(int a, int b) : (start=a; end=b)  Step 1: if start=1, increment start and print 1.  step 2: i=start  step 3: j=2  step 4: if i%j=0, increment count(initialised as 1) and jump to step 7.  Step 5: increment j.  step 6: if j<i/2, return to step 4.  step 7: if count>1, print i followed by 2 spaces.  Step 8: else print i followed by a ‘\*’ and a space.  Step 9: if (i-start+1)%10=0, begin a new line.  Step 10: Set count=1  Step 11: increment i  Step 12: if i<=end, return to step 3. |
| **FLOWCHART:** |  |
| **PROGRAM:** | #include<stdio.h>  void primes(int *start*, int *end*){      int count=1,x,offset,digits=1,y;      printf("the numbers in the range *%d* to *%d* with the primes starred are:\n",*start*,*end*);      x=*start*;      y=*start*;      offset=*start*%10;      while(y>=10){          y=y/10;          digits++;      }      for(int o=0;o<(offset-1)\*(2+digits);o++){          printf(" ");      }      if(*start*==1){  *start*++;          printf("1  ");      }      for(int i=*start*; i<=*end*; i++){          for(int j=2;j<i/2;j++){              if(i%j==0){count++;break;}          }          if(count>1){printf("*%d*  ",i);}          else{printf("*%d*\* ",i);}          if((i-x+offset)%10==0){printf("\n");}          count=1;      }  }  int main(){      int a,b,temp;      printf("Enter the range(positive numbers)\n");      scanf("*%d* *%d*",&a,&b);      if(a>b || b<1){printf("invalid input\n");return 0;}      if(a<1){a=1;}      primes(a,b);      return 0;  } |
| **RESULT:** | |
| **Program 2** | |
| **PROBLEM STATEMENT :** | Write a function which takes as parameters two positive integers and returns TRUE if the numbers are amicable and FALSE otherwise. A pair of numbers is said to be amicable if the sum of divisors of each of the numbers (excluding the no. itself) is equal to the other number. |
| **ALGORITHM:** | Step 1: START  Step 2: Read two numbers from input and store them in variables A and B.  Step 3: if amicable(A,B)=1, print that A and B are amicable.  Step 4: else print A and B are not amicable numbers.  Step 5: END  Algorithm for function amicable(int a,int b) (a and b are parameters and here, a=A,b=B)  Step 1: i=1, div=0  Step 2: if a%i=0, set div to div+i  Step 3: increment i  Step 4: if i<(a+2)/2, return to step 2.  Step 5: if div=b, return 1.  Step 6: else return 0. |
| **FLOWCHART:** |  |
| **PROGRAM:** | #include<stdio.h>  int amicable(int *a*, int *b*){      int div=0;      for(int i=1;i<(*a*+2)/2;i++){          if(*a*%i==0){              div=div+i;          }      }      if(div==*b*){return 1;}      else{return 0;}  }  int main(){      int a,b;      printf("Enter two numbers\n");      scanf("*%d* *%d*",&a,&b);      if(amicable(a,b))      printf("The numbers *%d* and *%d* are amicable\n",a,b);      else printf("The numbers *%d* and *%d* are not amicable\n",a,b);      return 0;  } |
| **RESULT:** | |
| **Program 3** | |
| **PROBLEM STATEMENT:** | Write a function to find the sum of the proper divisors of a given number ‘n’. The proper divisors of a number ‘n’ are the numbers less than n that divide it evenly. they do not include n itself. |
| **ALGORITHM:** | Step 1: START  Step 2: Read number from input and store it in A.  Step 3: div=divisors(A)  Step 4: print div  Step 5: END  Algorithm for function divisors(int a) (here a is a parameter and a=A)  Step 1: div=0, i=1  Step 2: if a%i=0, set div to div+i  Step 3: increment i  Step 4: if i<(a+2)/2, return to step 2  Step 5: return div |
| **FLOWCHART:** |  |
| **PROGRAM:** | #include<stdio.h>  int divisors(int *a*){      int div=0;      for(int i=1;i<(*a*+2)/2;i++){          if(*a*%i==0){              div=div+i;          }      }      return div;  }  int main(){      int a,div;      printf("Enter a number\n");      scanf("*%d*",&a);      div=divisors(a);      printf("The sum of the proper divisors of *%d* is *%d*\n",a,div);      return 0;  } |
| **RESULT:** | |
| **Program 4** | |
| **PROBLEM STATEMENT:** | The Mobius function M (N) is defined as  M (N) = 1         if N=1            = 0     if any prime factor is contained in N more than once            = (-1)   if N is the product of p different prime factors  Write a function MOBIUS as specified above. |
| **ALGORITHM:** | Step 1: START  Step 2: read a number from input and store it in A  Step 3: m=mobius(A)  Step 4: print m  Step 5: END  Algorithm for the function mobius(n) (here n is a parameter and n=A)  Step 1: if n=1, return 1 and terminate function.  Step 2: sqr=√n, count=1, i=2  Step 3: j=2  Step 4: if i%j=0, increment count and jump to step 8.  Step 5: increment j.  Step 6: if j<i/2, return to step 4.  Step 7: if count=1, if n%(i\*i)=0, return 0 and terminate function.  Step 8: set count=1  Step 9: increment i  Step 10: if i<=sqr, return to step 3  Step 11: return -1 and terminate function. |
| **FLOWCHART:** |  |
| **PROGRAM:** | #include<stdio.h>  #include<math.h>  int mobius(int *n*){      if(*n*==1){return 1;}      int sqr,count=1;      sqr=sqrt(*n*);      for(int i=2;i<=sqr;i++){          for(int j=2;j<i/2;j++){              if(i%j==0){count++;break;}          }          if(count==1){              if(*n*%(i\*i)==0){return 0;}          }          count=1;      }      return -1;    }  int main(){      int n,m;      printf("Enter a number\n");      scanf("*%d*",&n);      m=mobius(n);      printf("The value of M(*%d*) is (M(n) is mobius function): *%d*\n",n,m);  return 0;  } |
| **RESULT:** | |