|  |  |
| --- | --- |
| **Name** | **Hatim Yusuf Sawai** |
| **UID no.** | **2021300108** |
| **Experiment No.** | 3 |

|  |  |
| --- | --- |
| **AIM:** | 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.  Example:  Print a table as follows  1\* 2\* 3\* 4 5\* ... 10  11\* 12 13\* 14 15 ... 20 |
| **ALGORITHM:** | 1. START 2. Int Prime(int) 3. Main: 4. Input n 5. For i=1   p = prime(i)  if p==0  Ouput i\*  Else  Output i  If i%10==0  Output \n   1. Repeat step 5 till i<=n 2. STOP 3. START 4. Prime(int n): 5. If n==1   Return 1  Else  For i=2,i<=sqrt(n),i++  If n%i==0  Return 0   1. Return 0 2. STOP |
| **PROGRAM:** | #include <stdio.h>  #include <math.h>  int prime(int);  int main()  {  int n, p,i;  printf("Enter a number:\n");  scanf("%d", &n);  for(i=2;i<=n;i++)  {  p=prime(i);  if(p==0)  printf("%d\* ",i);  else  printf("%d ",i);    if(i%10==0)  printf("\n");  }  }  int prime(int n)  {  int i;  if (n == 1)  return 1;  else if (n > 1)  {  for (i = 2; i <= sqrt(n); i++)  {  if (n % i == 0)  return 1;  }  }  return 0;  } |
| **RESULT:** | |
| **Program 2** | |
| **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.  e.g. n=12 sum =1+2+3+4+6=16 |
| **ALGORITHM:** | 1. START (sum) 2. Sum=0 3. For i=1 4. If n%i==0 5. Sum +=i 6. Repeat steps 3-5 till i<n 7. Return sum 8. START (Main) 9. Input n 10. Output sum(n) 11. STOP |
| **FLOWCHART:** |  |
| **PROGRAM:** | #include<stdio.h>  int sum(int);  int main()  {  int n;  printf("Enter a number:\n");  scanf("%d",&n);  printf("Sum of Proper divisors = %d",sum(n));  return 0;  }  int sum(int n)  {  int i,sum=0;  for(i=1;i<n;i++)  {  if(n%i==0)  sum+=i;  }  return sum;  } |
| **RESULT:** | |
| **Program 3** | |
| **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. Ex. 1184 and 1210 are amicable. |
| **ALGORITHM:** | 1. START(Sum) 2. Sum=0 3. For i=1 4. If n%i==0 5. Sum +=i 6. Repeat steps 3-5 till i<n 7. Return sum 8. START(Amicable) 9. If sum(a)==b and sum(b)==a   Return 1   1. Return 0 2. START(Main) 3. Input a,b 4. If amicable(a,b) 5. Output Numbers are amicable 6. Else 7. Output Number are not amicable 8. STOP |
| **FLOWCHART:** |  |
| **PROGRAM:** | #include<stdio.h>  int sum(int);  int amicable(int,int);  int main()  {  int a,b;  printf("Enter two numbers:\n");  scanf("%d%d",&a,&b);  if(amicable(a,b))  printf("%d and %d are amicable numbers\n",a,b);  else  printf("%d and %d are not amicable numbers\n",a,b);  return 0;  }  int amicable(int a,int b)  {  if(sum(a)==b && sum(b)==a)  return 1;  return 0;  }  int sum(int n)  {  int i, sum = 0;  for (i = 1; i < n; i++)  {  if (n % i == 0)  sum += i;  }  return sum;  } |
| **RESULT:** | |
| **Program 4** | |
| **PROBLEM STATEMENT:** | Batch 3: A common method of evaluating powers is merely to perform repeated multiplications.A more efficient method of evaluating XN is:  Initialize PRODUCT to 1, POWER to X and M to N.  While M is non-zero repeat the following task:  If M is odd then  Replace PRODUCT by PRODUCT \* POWER  End If;  Replace M by M/2;  Replace POWER by POWER \* POWER  End task.  The required result is then in POWER.  Write a function which represents this procedure for calculating XN. |
| **ALGORITHM:** | 1. START (power) 2. p=1 3. while: 4. if n%2==0   p = p\*x   1. x = x\*x 2. n = n/2 3. Repeat steps 3-6 till n>0 4. Return p 5. START (Main) 6. Input x,n 7. Output x^n = power(x,n) 8. STOP |
| **PROGRAM:** | #include<stdio.h>  int power(int,int);  int main()  {  int x,n;  printf("Enter number and power:\n");  scanf("%d%d",&x,&n);  printf("%d^%d = %d",x,n,power(x,n));  return 0;  }  int power(int x,int n)  {  int p=1;  while(n>0)  {  if (n%2==1)  p=p\*x;  x=x\*x;  n=n/2;  }  return p;  } |
| **RESULT:** | |
| **CONCLUSION:** | In this experiment, we saw how to declare, define and use functions to introduce modularity in our programs thereby reducing time complexity and length of code. |