**Subject: PRF192- PFC**

**Workshop 02\_ Module and functions**

**Objectives:**

1. Practicing skills at analyzing and implementing programs using user-defined functions.
2. Making familiar with some basic algorithms

**PART 1:**

**PROGRAM 1**: Write a function that prints all numbers from 1 to 100 as below:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

**PROGRAM 2**: Write a function to print all the numbers from 1 to 100, which are divisible by 3 and divisible by 5.

**Program 3**: Write a program in C to find the square of any number using the function.    
Test Data :  
Input any number for square : 20  
*Program pected Output* :

The square of 20 is : 400.00

**Program 4:** Write a program in C to swap two numbers using function.    
Test Data :  
Input 1st number : 2  
Input 2nd number : 4  
Program pected Output :

Before swapping: n1 = 2, n2 = 4

After swapping: n1 = 4, n2 = 2

**Program 5:** Write a program in C to check a given number is even or odd using the function.    
Test Data :  
Input any number : 5  
Program pected Output :

The entered number is odd.

**PROGRAM 6**: Write a program in C to find the sum of the series 1!/1+2!/2+3!/3+4!/4+5!/5 using the function.    
Program pected Output :

The sum of the series is : 34

**PROGRAM 7**: Write a program in C to convert decimal number to binary number using the function.    
Test Data :  
Input any decimal number : 65  
Program pected Output :

The Binary value is : 1000001

**PROGRAM 8:** Write a program in C to check whether a number is a prime number or not using the function.   
Test Data :  
Input a positive number : 5  
Program pected Output :

The number 5 is a prime number.

**PROGRAM 9:** Write a program in C to print all perfect numbers in given range using the function.    
Test Data :  
Input lowest search limit of perfect numbers : 1  
Input lowest search limit of perfect numbers : 100  
Program pected Output :

The perfect numbers between 1 to 100 are :

6 28

**PART 2:**

**Program 1:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | **Definition**: A prime is positive integer that is greater than 1 **and** it is the multiple of 1 and itself only.  **Theorem**: The integer n is a prime if and only if n>1 and it can not be divided by all integers from 2 to ⎣ square root of n⎦ .  Use the library **math.h** to get the function **sqrt(double)** for getting the square root of a positive number. |
| **Problem** | Write a C that will accept a positive integer n, n>=2 then print out primes between 2 and n. |
| **Analysis**  *Nouns:*  *positive integer*  *🡪 int n* | **Suggested algorithm (logical order of verbs)**  Begin  Do {  Accept n;  }  While ( n<2);  For ( i=2 to n )  If (**i is a prime** ) Print out i; 🡪 **Function int prime (int i)**  End |
| **Algorithm for checking whether an integer is a prime or not** | **int prime( int n ) {**  **int m = sqrt(n);** /\* m: square root of n \*/  **int i;**  /\* variable having value from 2 to m \*/  **if (n<2) return 0;** /\* Condition 1 is not satisfied \*/  **for ( i=2; i<=m; i++)** /\* checking the second condition \*/  **if (n%i==0) return 0 ;** /\* n is divided by i 🡪 n is not a prime \*/  **return 1;** /\* n is a prime \*/  **}** |

**Program 2:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | Leap year (y): (y%400==0 || ( y%4==0 && y%100!=0)) |
| **Problem** | Write a C program that will accept data of a day then print out whether they are valid or not. |
| **Analysis**  Data of a day   * int d, m, y | **Suggested algorithm (logical order of verbs)**  Begin  Accept d, m, y  If (valid(d,m,y)) print out “valid date”  Else print out “invalid date”  End |
| **Algorithm for checking whether a date is valid or not** | **int validDate ( int d, int m, int y)** {  **int** maxd = 31; /\*max day of months 1, 3, 5, 7, 8, 10, 12 \*/  /\* basic checking \*/  **if** ( d<1 || d>31 || m<1 || m>12) **return** 0;  /\* update maxd of a month \*/  **if** ( m==4 || m==6 || m==9 || m=11) maxd=30;  **else if** (m==2) {  /\* leap year? \*/  **if** ( y%400==0 || ( y%4==0 && y%100!=0) maxd=29;  **else** maxd=28;  }  return d<=maxd;  } |

**Program 3:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | A point p is in a circle if the distance from the center to p is less than the radius. |
| **Problem** | Write a C program that will accept a point and a circle having the center is (0,0) then print out the relative position of this point with the circle. |
| **Analysis**  Nouns:  A point 🡪 double x,y  A circle 🡪 double r  Relative position   * int result * -1: (x,y) is out of the circle * 0: (x,y) is on the circle * 1: (x,y) is in the circle | **Suggested algorithm (logical order of verbs)**  Begin  Accept x, y;  Do {  Accept r;  }  While(r<0);  result = getRelPos(x,y,r);  if (result ==1) Print out “The point is in the circle”;  else if (result==0) Print out “The point is on the circle”;  else Print out “The point is out of the circle”;  End |
| **Algorithm for**  **getting relative position of a point with a circle** | **int getRelPos ( double x, double y, double r) {**  **double d2=x\*x + y\*y;** /\* d2= x2+ y2 \*/  **double r2= r\*r;** /\* r2\*/  **if (d2<r2) return 1 ;** /\* d2<r2 🡪the point is in the circle \*/  **else if (d2==r2) return 0 ;** /\* d2=r2 🡪the point is on the circle \*/  **return -1 ;** /\* d2 > r2 🡪the point is out of the circle \*/  **}** |

**Program 4:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | **n! = 1\*2\*3\*…\*n** |
| **Problem** | Write a C program that will accept a positive integer then print out its factorial. |
| **Analysis**  A positive integer   * int n | **Suggested algorithm (logical order of verbs)**  Begin  Do {  Accept n;  }  While (n<0);  Print out factorial(n);  End. |
| **Algorithm for**  **Computing factorial of an integer** | **double factorial ( int n) {**  double p=1;  int i;  for (i=2; i<=n; i++) p \*= i;  return p;  **}** |

**Program 5:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | Fibonacci sequence: 1 1 2 3 5 8 13 21 34 …  Two first numbers: 1  Others: Its value is the sum of 2 previous numbers |
| **Problem** | Write a C program that will print out the value at the nth position in Fibonacci sequence. |
| **Analysis**  A position   * int n | **Suggested algorithm (logical order of verbs)**  Begin  Do {  Accept n;  }  While (n<1);  Print out fibo(n);  End. |
| **Algorithm for**  **Computing the nth value of the Fibonacci sequence** | **double fibo ( int n) {**  **int t1=1, t2=1, f=1, i ;**  **for ( i= 3, i<=n; i++) {**  **f= t1 + t2;**  **t1= t2;**  **t2=f;**  **}**  **return f;**  **}** |

**How to compute the nth value of the Fibonacci sequence**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Position 1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **1** | **1** | **2** | **3** | **5** | **8** | **13** | **21** | **34** | **55** |
| T1 | T2 | F |  |  |  |  |  |  |  |
|  | T1 | T2 | F |  |  |  |  |  |  |
|  |  | T1 | T2 | F |  |  |  |  |  |
|  |  |  | T1 | T2 | F |  |  |  |  |
|  |  |  |  | T1 | T2 | F |  |  |  |
|  |  |  |  |  | T1 | T2 | F |  |  |
|  |  |  |  |  |  | T1 | T2 | F | … |

**Program 6:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** |  |
| **Problem** | Write a C program that will accept a positive integer then print out whether it is an element of the Fibonacci sequence or not. |
| **Analysis**  An integer 🡪 int n | **Suggested algorithm (logical order of verbs)**  Begin  Do {  Accept n;  }  While (n<1);  If ( isFibonacci(n)==1) Print out “It is a Fibonacci element.”;  Else print out “It is not a Fibonacci element.”  End |
| **Algorithm for**  **Checking whether an integer is a element of the Fibonacci sequence or not** | **int isFibonacci (int n)**  { int t1=1, t2=1, f=1;  if (n==1) return 1; /\* n belongs to the Fibonacci sequence\*/  while (f<n) /\* Find out the Fibo number f to n \*/  { f= t1 + t2;  t1=t2;  t2=f;  }  return n==f; /\* if n==f 🡪 n is Fibo element 🡪 return 1 \*/  } |

**Program 7:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | Getting the rightmost digit of the integer n: **n%10** |
| **Problem** | Write a C program that will carry out some times. In each time, a nonnegative integer is accepted then print out the sum of its decimal digits. The program will terminate when its value of accepted number is negative. |
| **Analysis**  Sum 🡪 int S=0  Accepted integer   * int n | **Suggested algorithm (logical order of verbs)**  Begin  Do  { Accept n;  If (n>=0)  { S = sumDigits(n);  Print out S;  }  }  While (n>=0);  End |
| **Algorithm for**  **Computing sum of digits of a nonnegative integer** | **int sumDigits (int n)**  **{** int sum=0; /\* initialize sum of digits \*/  Do  { int remainder = n%10 ; /\* Get a digit at unit position \*/  n = n/10;  sum += remainder;  }  while (n>0);  return sum;  **}** |

**Program 8:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | **Making a real number from its integral part and its fraction** (its fraction must be positive).  Program ample : 32 25 🡺 32**.**25  25 🡺 0.25 🡺 32+0.25= 32.25  Program ample -51 139 🡺 -51**.**139  139 🡺 0.139 🡺 -51- 0.139= -51.139  **double makeDouble(int ipart, int fraction)**  { double d\_f= fraction;  while (d\_f >=1) d\_f = d\_f/10; /\* create the fraction <1 \*/  if (ipart<0) return ipart – d\_f; /\* case -51 – 0.139 \*/  return ipart + d\_f ; /\* case 32 + 0.25 \*/  } |
| **Problem** | Write a C program that will accept the integral part and fraction of a real number then print out the this real number. |
| **Analysis**  Integral part   * int ipart   Fraction   * int fraction   Real number   * double value | **Suggested algorithm (logical order of verbs)**  Begin  Accept ipart;  Do  { Accept fraction;  }  While fraction<0;  value= makeDouble(ipart,fraction);  Print out value;  End |

**Program 9:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | Find out the greatest common divisor **(gcd)** and least common multiple **(lcm)** of two positive integers:  *Find out gcd of a and b*  a b a b  14 21 13 8  14 7 5 8  7 7 5 3  2 3  2 1  1 1  **int gcd( int a, int b)**  { while ( a != b )  if a>b then a -=b;  else b -= a;  return a;  }  **int lcm ( int a, int b)**  { return a\*b/ gcd(a,b);  } |
| **Problem** | Write a C program that will accept two positive integers then print out their greatest common divisor and least common multiple. |
| **Analysis**  Two integers   * int a, b   gcd 🡪 int d  lcm 🡪 int m | **Suggested algorithm (logical order of verbs)**  Begin  Do  { Accept a, b;  }  While ( a<=0 OR b <=0);  d = gcd(a,b);  m = lcm (a.b);  Print out d;  Print out m;  End |

**Program 10:**

|  |  |
| --- | --- |
| **Objectives** | Practice implementing simple functions |
| **Related knowledge** | **Print out the minimum and the maximum digits of a nonnegative integer integer**  Program ample: n= 10293 🡺 Print out 9, 0  **void print****MinMaxDigits( int n)**  { int digit; /\* Variable for Program tracting 1 digit \*/  int min, max ; /\* Result variables \*/  digit = n% 10; /\* get the first rightmost digit: 3 \*/  n=n/10; /\* 1029, the remainder needs to proceed after\*/  min=max=remainder; /\* initialize results \*/  while (n>0)  { digit = n%10; /\* Get the nProgram t digit \*/  n=n/10;  if (min > remainder) min=remainder; /\* update results \*/  if (max < remainder) max=remainder;  }  Print out min, max;  } |
| **Problem** | Write a C program that will accept a non-negative integer then print out its minimum and maximum digits. |
| **Analysis**  Noun:  A integer   * int n | **Suggested algorithm (logical order of verbs)**  Begin  Do  { Accept n;  printMinMaxDigits(n);  }  While (n<0);  End |