**Shubhan Singh**

**SE-Comps B/Batch C**

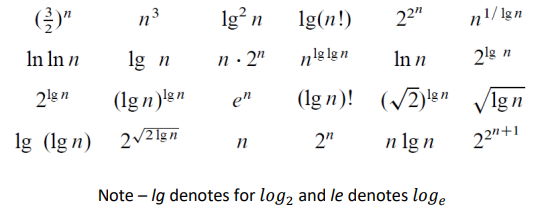
**2022300118**

DAA Experiment 0

**Aim** – To implement the various functions e.g. linear, non-linear, quadratic, exponential etc.

**Details** – A function is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output. Let A & B be any two non-empty sets. mapping from A to B will be a function only when every element in set A has one end, only one image in set B.

**Problem Definition & Assumptions** – For this experiment, you must implement at least 10 functions from the following list.



The input (i.e. n) to all the above functions varies from 0 to 100 with increment of 1. Then add the function n! in the list and execute the same for n from 0 to 20.

**Source code(C language):**

* I have printed the output in a csv file so that the data can be easily imported into excel. For this purpose, the values returned by the functions are not returned immediately, and are instead stored in a 2-D array, so that they can be properly formatted.
* Function pointers have been used to reduce the size of code as they allow to traverse an array of functions pointers in a loop, rather than calling each one separately.
* The 10 chosen functions (apart from factorial) are:

,,,,,, ,,,

**#include<stdio.h>**

**#include<math.h>**

**#include<stdlib.h>**

**long double fastpow(long double a, int b){//faster pow function when index is integer**

**long double ans=1,power=a;**

**while(b>0){**

**if(b%2==1){**

**ans\*=power;**

**}**

**power\*=power;**

**b/=2;**

**}**

**return ans;**

**}**

**// all functions have long double return type and all attributes have long double data type**

**// to allow for use of array of function pointers**

**long double x(long double x){**

**return x;**

**}**

**long double x3(long double x){**

**return x\*x\*x;**

**}**

**long double lgx(long double x){**

**return log2(x);**

**}**

**long double lnx(long double x){**

**return log(x);**

**}**

**long double lg\_sq\_x(long double x){**

**return lgx(x)\*lgx(x);**

**}**

**long double ln\_lnx(long double x){**

**return lnx(lnx(x));**

**}**

**long double x\_lgx(long double x){**

**return x\*lgx(x);**

**}**

**long double x\_pow\_1bylgx(long double x){**

**return pow(x,pow(lgx(x),-1));**

**}**

**long double two\_x(long double x){**

**return fastpow(2,(int)x);**

**}**

**long double e\_x(long double x){**

**return fastpow(exp(1),(int)x);//using inbuilt function for accurate value of e**

**}**

**long double factorial(long double x){**

**if(x==0){**

**return 1;**

**}**

**long double ans=1;**

**for(int i=x;i>1;i--){**

**ans\*=i;**

**}**

**return ans;**

**}**

**int main(){**

**FILE\* file=fopen("Exp\_1\_A.csv","w");//using csv formatting so file is easily opeened in excel**

**if(file==NULL){//checking if fopen worked**

**printf("Failed to open file\n");**

**}**

**//i stored all values in an array first so that they can be printed side by side into the file**

**long double \*\*arr=malloc(11\*sizeof(long double \*));//initializing 11 1D arrays to store data for the 10 functions+factorial**

**for(int i=0;i<10;i++){**

**arr[i]=malloc(101\*sizeof(long double));**

**}**

**arr[10]=malloc(21\*sizeof(long double));//array of size 20 for factorial**

**long double (\*fnptr\_arr[])(long double)={x,x3,lgx,lnx,lg\_sq\_x,ln\_lnx,x\_lgx,x\_pow\_1bylgx,two\_x,e\_x,factorial};**

**//using an array of function pointers so all functions can be called in a loop**

**for(int i=0;i<10;i++){**

**for(int n=0;n<=100;n++){**

**arr[i][n]=fnptr\_arr[i](n);**

**}**

**}**

**for(int n=0;n<=20;n++){**

**arr[10][n]=fnptr\_arr[10](n);**

**}**

**//in the csv file format, contents of each cell are seperated by commas, empty cells are represented by consecutive**

**//commas, and newlines by newline characters**

**fprintf(file,"x,x,x^3,lg(x),ln(x),lg^2(x),ln(ln(x)),xlg(x),x^(1/lg(x)),2^x,e^x,x!\n");**

**for(int i=0;i<=20;i++){**

**fprintf(file,"%d,",i);**

**for(int j=0;j<11;j++){**

**fprintf(file,"%.5Lf,",arr[j][i]);**

**}**

**fprintf(file,"\n");**

**}**

**for(int i=21;i<=100;i++){**

**fprintf(file,"%d,",i);**

**for(int j=0;j<10;j++){**

**fprintf(file,"%.5Lf,",arr[j][i]);**

**}**

**fprintf(file,"\n");**

**}**

**fclose(file);**

**for(int i=0;i<11;i++){**

**free(arr[i]);**

**}**

**free(arr);**

**return 0;**

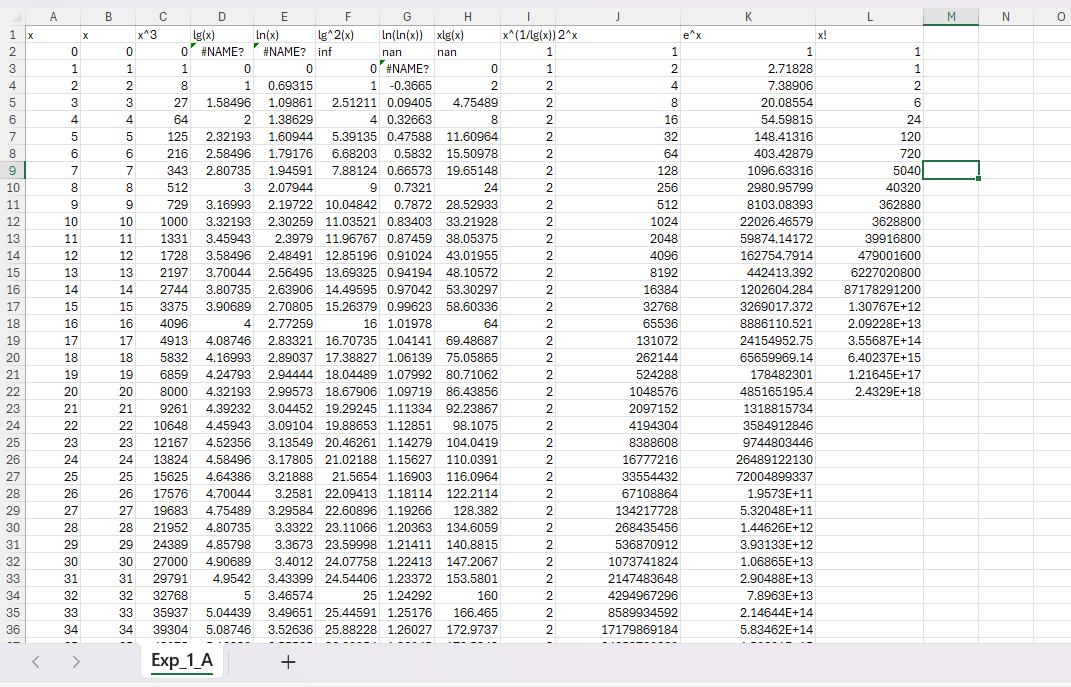
**}**

**Output:**

**A screen shot of a computer screen

Description automatically generated**

File created



(#NAME? refers to -inf)

CSV file

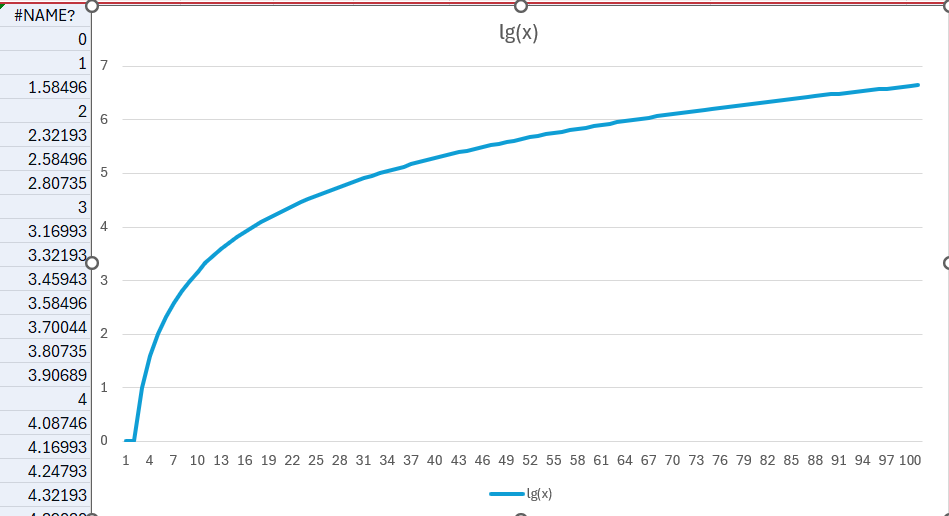
**PLOTS:**

**A graph with a red line

Description automatically generated**

A graph with a green line

Description automatically generated



(#NAME? refers to -infinity)

A graph with a line

Description automatically generated

**A graph with a green line

Description automatically generated**

**A graph with a line

Description automatically generated**

**A graph with a red line

Description automatically generated**

**A graph with a line and numbers

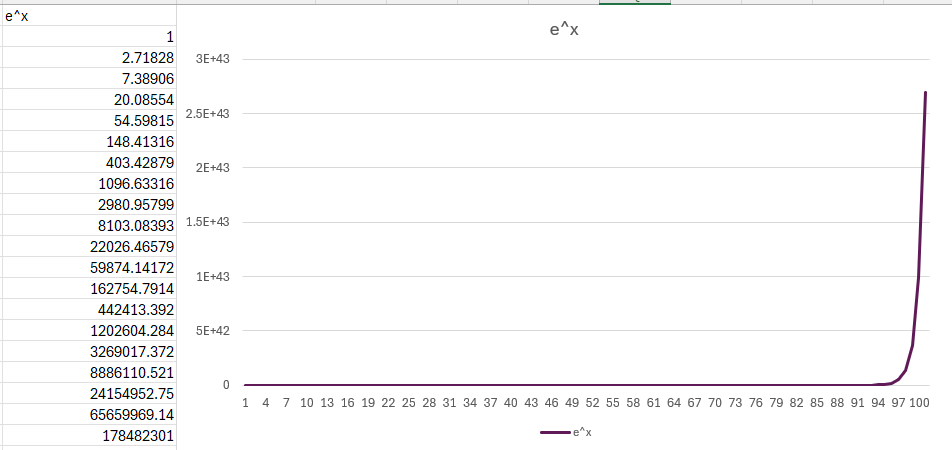
Description automatically generated**

A white screen with numbers and a number on it

Description automatically generated with medium confidence (true scale)

A graph with a line

Description automatically generated (Logarithmic scale)

 (True scale)

A graph with a purple line

Description automatically generated (Logarithmic scale)

1. **n!**

A screenshot of a graph

Description automatically generated (True scale)

A graph with a green line

Description automatically generated (Logarithmic)

Algorithm:

Here's the pseudocode for the functions code:

1. Function fastpow(a, b):
   * Initialize ans to 1 and power to a.
   * While b is greater than 0:
     + If b is odd, multiply ans by power.
     + Square power.
     + Divide b by 2.
   * Return ans.
2. Function x(x):
   * Return x.
3. Function x3(x):
   * Return x cubed.
4. Function lgx(x):
   * Return the base-2 logarithm of x.
5. Function lnx(x):
   * Return the natural logarithm of x.
6. Function lg\_sq\_x(x):
   * Return the square of the base-2 logarithm of x.
7. Function ln\_lnx(x):
   * Return the natural logarithm of the natural logarithm of x.
8. Function x\_lgx(x):
   * Return x times the base-2 logarithm of x.
9. Function x\_pow\_1bylgx(x):
   * Return x raised to the power of the reciprocal of the base-2 logarithm of x.
10. Function two\_x(x):

* Return 2 raised to the power of x.

1. Function e\_x(x):

* Return e (the base of the natural logarithm) raised to the power of x.

1. Function factorial(x):

* If x is 0, return 1.
* Initialize ans to 1.
* For each integer i from x down to 2, multiply ans by i.
* Return ans.

1. In main():

* Open a file named "Exp\_1\_A.csv" for writing.
* If the file failed to open, print an error message and exit.
* Allocate memory for an 11-element array of pointers to long double arrays.
* For each of the first 10 elements of the array, allocate memory for a 101-element long double array.
* For the 11th element of the array, allocate memory for a 21-element long double array.
* Initialize an array of function pointers with the functions defined earlier.
* For each of the first 10 functions and each integer n from 0 to 100, call the function with n as the argument and store the result in the corresponding element of the long double array.
* For the 11th function and each integer n from 0 to 20, call the function with n as the argument and store the result in the corresponding element of the long double array.
* Write the headers to the CSV file.
* For each integer i from 0 to 20, write i and the first 11 elements of the ith row of the long double array to the CSV file.
* For each integer i from 21 to 100, write i and the first 10 elements of the ith row of the long double array to the CSV file.
* Close the file.
* Free the memory allocated for the long double arrays.
* Free the memory allocated for the array of pointers.

**Theory:**

1. Function pointers in C:

Function pointers in C are pointers that point to functions, instead of pointing to data like integers, characters, or structures. They can be used to pass functions as parameters to other functions, or to reference a function in a data structure.

1. Long double data type in C:

The long double data type in C is a floating point data type that is more precise than the float and double data types. It is used when higher precision is required.

The size of long double can vary between different platforms and compilers, and varies from 10 to 16 bytes, though it is 10 byles on GNU C compilers. Its type specifier is %Ld.

1. File handling:

* **File Pointer**: In C programming, a file pointer is a special kind of pointer used for interacting with files. It's essentially a pointer to a structure of type FILE that contains information about the file, such as its name, its status (whether it's open or closed), its current position, and more. A file pointer is used as an argument to various file handling functions to perform operations like reading from or writing to a file.
* **fopen**: The fopen function is used to open a file and associate it with a file pointer. It takes two arguments: the name of the file to be opened and the mode in which to open it. The mode can be for reading ("r"), writing ("w"), appending ("a"), and others. If the file is successfully opened, fopen returns a pointer to the FILE structure associated with the file. If the file cannot be opened (for example, if the file does not exist and the mode is "r"), fopen returns NULL.
* **fprintf**: The fprintf function is used to write formatted output to a file. It works similarly to the printf function, but instead of writing to the console, it writes to the file associated with the provided file pointer. The first argument to fprintf is the file pointer, followed by a format string (which can include format specifiers like %d, %s, etc.), and then the values to be formatted and written. The function returns the number of characters written, or a negative value if an error occurs.