UNIT 18 LIST OF PRACTICAL SESSIONS

In this unit, we list the practical sessions for this course. While it is compulsory to complete those sessions which are marked compulsory, we encourage you to do as many sessions as possible. Apart from the practical sessions, we advise you to compile and run the examples in the Block to get more practice. While we have checked all the programs in the blocks for errors, it is quite possible that some of them still have errors in them. If you find any, please send a mail to the course coordinator pointing the mistake.

For the programs based on numerical analysis, you can refer to the blocks of MTE-10, Numerical Analysis, which is a part of the IGNOU Bachelor's degree program. The blocks of this course will be available at your program study centre.

Add comments in the program and in functions, documenting what the program or function does. Indent your programs properly so that anyone can read them easily and understand them. If you are working on linux, the indent program may be available on your computer. For example, if you want to indent your program in the style followed in the book by Kernighan and Ritchie, you can invoke indent using the command

```
indent -kr file_name.c
```

Indent offers many other options. Read the documentation for the program if necessary.

Also, write a friendly interface, wherever required, for the user to interact with the program. Your program should also check if the input is appropriate and print an error if it is not.

Session 1(Compulsory)

The aim of this session and the next session is to familiarise you with the process of compiling and running C programs. The example programs for the first three sessions are from the first Block, so take it to the study center with you when you go for practicals.

Program 1

1. Compile and run the program unit1-prog1.c given in page 9 of Block 1 from the command line.
2. Open the file unit1-prog1.c in you IDE and compile and run the file. (You have to add the line `getchar();` before the return statement if you are using Dev-C++.)
3. Change the program so that it prints

   ```
   This is very easy!!
   ```

Program 2

1. Compile and run the program unit1-prog8.c in page 11 of Block 1.
2. Change the values of x and y to x = 12, y = 13 and x = 11, y = 14 and get the output.
3. Try giving decimal values x = 1.2, y = 4.1 and x = 0.2 and y = 5 and study the output. How far is your answer correct?

Program 3

1. Calculate the integral $\int_0^1 \frac{1}{1+x^2}\,dx$ by actual integration.
2. Compile and run the program unit1-simpsonf.c in page 17 of Block 1 and compare the values you get.
3. Change the function to a) x^2 b) e^x. (See the list of functions given in Table 1, page 16, Block 2.) Again, compare the values given by the program with the values you get by actual integration. Note down the difference.

Session 2(Compulsory)

Program 1

Insert `printf()` statements in unit3-prog1.c in page 53, Block 1, compile it and run it to check your answer.

Program 2

Compile and run the file unit3-prog2.c in page 54 of Block 2 to check your answer for the Exercise E3) of Unit 3.

Program 3

Complete the program unit3-prog0.c in page 71 of Block 1, which is given as the solution to exercise E1) of Unit 3, by giving values for the remaining 3 cases. Compile and run the program and check the output.

Program 4

In unit3-prog0.c, instead of including the values in the program itself, modify it so that it reads the values of a and b using `scanf()`. Compile and run the program. Give values for a and b covering the 4 cases and check your output.

Program 5

Create a program that prompts the user for basic pay and calculates and prints basic + HRA(30% of basic)+DA(46% of basic).

Session 3(Compulsory)

Program 1

Write a program similar to unit2-ex7ans.c in page 45 of Block 1 that checks for overflow in multiplication when the two operands are **unsigned long** and when they are **long double**. The program should also print the overflow when it occurs.

Program 2

Write a program that prompts for a natural number n and prints the nth Fibonacci number. It should also print an error message if the Fibonacci number is bigger than `ULONG_MAX`.

Program 3

Write a program that finds all primes less than ULONG_MAX and saves it in a file called primes. You can modify any of the programs we have given in Block 2 for finding primes.

Session 4

Write a C function to determine the number of days between two dates passed to it. (Hint: Find the number of days, say n, between the first date and January 1, 1900; find the number of days, say m, between the second date and January 1, 1900. Your program should return the absolute value of the differences $n - m$.)

Session 5(Compulsory)

Program 1

Write a function that evaluates the polynomial by Horner's method. Horner's method is as follows(See MTE-10 material if necessary.): If
$f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, the Horner's method is given by the equation

$$f(x) = ((\cdots((a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0)$$

It should prompt for the coefficients of the polynomial, store the values in an array, prompt the value for which the polynomial is to be evaluated and evaluate and print the value. Your program should be able to evaluate any polynomial of degree less than 20.

Program 2

If $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, define

$$f_o = \sum_{\substack{0 \le i \le n \\ i \text{ odd}}} a_i x^i, f_e(x) = \sum_{\substack{0 \le i \le n \\ i \text{ even}}} a_i x^i$$

Write functions similar to Horner's method for evaluating $f_o(x)$ and $f_e(x)$. Note that $f(a) = f_e(a) + f_o(a)$ and $f(-a) = f_e(a) - f_o(a)$. It is easy to show that, if the polynomial $f(x)$ has integer coefficients and has a rational root $\frac{p}{q}$, then $p \mid a_0$ and $q \mid a_n$. In particular, it has an integer root a, then $a \mid a_0$. Using this fact and the functions for finding $f_o(a)$ and $f_e(a)$ for any a, write a function for checking whether a polynomial $f(x)$ has integer roots by finding the value of $f(\pm a)$ for all divisors a of a_0.

Program 3

Write a program that computes the value of $\tan^{-1} x$ using the power series expansion

$$x - \frac{x^3}{3} + \frac{x^5}{5} - \cdots$$

Compare the values obtained for $\frac{1}{2}$ and $\frac{1}{\sqrt{3}}$ with the values obtained from a table or a scientific calculator. Write a program to find the value of π using the relation
$\pi = 16\tan^{-1}\left(\frac{1}{5}\right) - 4\tan^{-1}\left(\frac{1}{239}\right)$

Session 6

Write a program that uses Simpson's one third rule to evaluate definite integrals of function for arbitrary number of intervals. Your program should prompt for the upper

and lower limits of integration and the number of subintervals and evaluate the integral. It should check if the number of intervals is even. Use your program to evaluate the integrals $\int_0^{\pi/2} \sqrt{\sin x}\, dx$ and $\int_0^{1/3} \frac{1}{\sqrt{1-x^2}}\, dx$ for the number of intervals $n = 10$, $n = 50$, $n = 100$. Compare the values with trapezoidal rule with the values you get from the Simpson's rule.

Session 7(Compulsory)

Program 1

Write a function that solves equations by bisection method. Use it to find a solution of
a) $x^2 - 5x - 24 = 0$ b) $x\log_{10} = 1.2$

Program 2

Write a program that solves algebraic equations by false position(regula falsi) method. Use it to solve the equations in Program 1 above.

Session 8(Compulsory)

Write a function that solves an intermediate value problem by Runge-Kutta method of $O\left(h^4\right)$ for different values of h. Your program should prompt for the value of h, find the number of iterations required and solve the problem. Use your program to solve the following problems:

1. $y' = t + y$, $y(0) = 1$ for $t = 0.5$ with $h = 0.1$, $h = 0.05$ and $h = 0.01$.
2. $y' = 2y/t$, $y(1) = 2$ for $t = 2$ with $h = 0.1$, $h = 0.05$ and $h = 0.01$.

Session 9(Compulsory)

Program 1

In this program you will create a menu using **switch()** statement. Write a currency converter program that converts rupee to dollar, yen or euro or converts any of these currencies to rupee. When the program is run it should print a menu as follows:

```
What do you want to do?

  1. Convert rupee to other currencies.
  2. Convert other currencies to rupee.

Press 1 or 2 to make your choice.  Press any other key to
exit.
```

If the user presses 1, for example, it should display the following menu:

```
Convert rupee to

  1. Dollar
  2. Yen
  3. Euro
```

Press 1, 2 or 3 to make your choice. Press u to go to the previous menu.

After the user presses 1, 2 or 3, the program should scan the amount in rupee, make the conversion and print the value. It should once again show the first menu. If the user presses any key other than 1, 2 or 3, it should print an error message and ask the user to make a correct choice and print the menu again. The user should be able to exit from the program by pressing any key other than 1 or 2 in the first menu.

Program 2

Consider the problem of estimating the number of real and imaginary roots of a polynomial equation with real coefficients.

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0 \quad (1)$$

We say that a polynomial has a **change of sign** at a particular term if the sign of that term is different from the next higher degree term. We say that a polynomial has **continuation** at a particular term if the sign of the term is the same as the next higher degree term. The equation is **complete** if no coefficient is zero. If $a_r = 0$, we say that r^{th} term is **missing**. For example, in the polynomial $5x^6 - 3x^4 - 4x^3 + 5x^2 - 3x + 1 = 0$ has changes of sign at the terms $-3x^4$, $5x^2$, $-3x$ and 1. It has a continuation at $-4x^3$. The x^5 term is missing.

Recall the **Descartes Rule of Signs** for finding the number of positive and negative roots of a polynomial with real coefficients.

Theorem 1 (Descartes' rule of signs) *The equation* $f(x) = 0$ *cannot have more positive roots than* $f(x)$ *has changes of sign or more negative roots than* $f(-x)$ *has changes of sign.*

Write a C program that prompts for the coefficients of a polynomial and prints the maximum possible number of real roots. Use it to find the maximum number of real roots of the polynomials.

Session 10(Compulsory)

Program 1

In this program you will write function that uses a more efficient method of powering an element in a group. The basic idea behind the algorithm is the fact if $n = \sum_{i=0}^{k} a_i 2^i$ where each a_i is 0 or 1, $a^n = \prod_{a_i=1} a^{2^i}$. Here is the algorithm(**Source**: *A Course in Computational Algebraic Number Theory* by *Henri Cohen*, GTM 138, published by Springer.):

Algorithm(Binary Powering): Given $a \in G$ and $n \in \mathbf{Z}$, this algorithm computes a^n in G. We write 1 for the identity element in G. In the algorithm we write $\lfloor x \rfloor$ for the largest integer smaller than x. For example $\lfloor 3.2 \rfloor = 3$

1. [Initialise] Set $y \rightarrow 1$. If $n = 0$ output y and terminate. If $n < 0$, let $N \leftarrow -n$ and $z = g^{-1}$. Otherwise, set $N \leftarrow n$ and $z \leftarrow g$.
2. [Multiply?] If N is odd set $y \leftarrow z \cdot y$.
3. [Halve N] Set $N \leftarrow \lfloor N/2 \rfloor$. If $N = 0$, output y as the answer and terminate the algorithm. Otherwise, set $z \leftarrow z \cdot z$ and go to step 2.

Work through the algorithm for $a = 3$ and $n = 5$ manually to understand the algorithm. Then, write a function that implements the algorithm for the group $\left(\frac{\mathbf{Z}}{p\mathbf{Z}}\right)^*$, p a prime, for positive values of n only.

Program 2

Write a program that finds ord(a), the order of $a \in \frac{\mathbf{Z}}{p\mathbf{Z}}$ for a prime p. The program should first factorise $p-1$ using the already computed list of primes in Session 3, say $p-1 = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$. It should find ord(a) by finding the power of each p_i dividing ord(a). For example, to find the order of p_1 dividing ord(a) the program should proceed as follows: First, find $b = a^{\frac{p-1}{p^{n_1}}}$. Then, find m_1 such that $b^{p^{m_1}} = 1$ and $b^{p^{m_1-1}} \neq 1$. The power of p_1 dividing ord(a) is $p_1^{m_1}$.

Use your program to find the order of 768462011 in $\frac{\mathbf{Z}}{p\mathbf{Z}}$ for $p = 4264967269$. Use the Binary Powering algorithm discussed in Program 1 of this session for taking powers.

Session 11(Compulsory)

Program 1

Write a programs for linear regression. Given the values x_i, $y_i = f(x_i)$, $1 \leq i \leq n$, you have to fit a straight line $Y = aX + b$ for the data. The values of a and b are given by

$$b = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}, a = \frac{\sum y_i}{n} - b\frac{\sum x_i}{n}$$

Write a program that reads value n, x_is and y_is and outputs a and b. Use the program to fit a line for the following data.

x	1	2	3	4	5
y	3	4	5	6	8

Program 2

Write a program to fit a transcendental curve of the form $Y = aX^b$ for a given set of data. We have

$$b = \frac{n\sum \ln x_i \ln y_i - \sum \ln x_i \sum \ln y_i}{n\sum (\ln x_i)^2 - (\sum \ln x_i)^2}, \ln a = R = \frac{\sum \ln y_i}{n} - b\frac{\sum \ln x_i}{n}$$

and $a = e^R$. Use these formulae to fit a curve of the form $y = ax^b$ for the date given below:

x	1	2	3	4	5
y	0.5	2	4.5	8	12.5

Session 12(Compulsory)

In this session, you will create a $n \times n$ array of **float** using `calloc()`. Here are the functions for creating and destroying a $n \times n$ square matrix of **float**.

```
float **create_square_matrix(int n)
{
  float **matrix = calloc(n, sizeof(float *));
  int i;
  if (!matrix) {
    fprintf(stderr, "fatal error\ninsufficient memory\n");
    fprintf(stderr, "from create_matrix\n");
    exit(1);
```

```
  }
  for (i = 0; i < n; i++) {
    matrix[i] = calloc(n, sizeof(float));
    if (!matrix[i]) {
      fprintf(stderr, "fatal error\ninsufficient memory\n");
      fprintf(stderr, "from create_matrix\n");
      exit(1);
    }
  }
  return matrix;
}
void delete_square_matrix(int n, float **matrix)
{
  int i;
  for (i = 0; i < n; i++)
      free(matrix[i]);
  free(matrix);
}
```

Use the matrix for creating matrix to create a 3×3 matrix of floats and set `m[i][j]=(i+j)/2.0`. Print the matrix, then call the delete matrix function and exit the program. Write a program that solves linear equations by partial pivoting. Note that you can interchange rows by using a pointer called `temp` defined as **float** `*temp`. Instead interchanging element by element, you can just swap pointers using `temp`. Use your program to solve the set of equations

$$\begin{aligned} 0.232x_1 + 0.457x_2 + x_3 &= 0.248 \\ 0.221x_1 + 0.129x_2 + 0.126x_3 &= 0.124 \\ 0.127x_1 + 0.328x_2 + 0.192x_3 &= 2.341 \end{aligned}$$

Session 13

Write a program for finding the inverse of a matrix using Gauss-Jordan method. Use arrays created using `calloc()` as in the previous session. Use your program to find the inverse of the matrix

$$\begin{pmatrix} 2 & 5 & 4 \\ 5 & 8 & 5 \\ 4 & 5 & 4 \end{pmatrix}$$

Session 14(Compulsory)

Define a **struct** for working with natural numbers by

```
typedef struct RAT{
long      num;
unsigned long  denom;
}rat;
```

Here, we assume that the denominator is always positive. Write functions for:

1. Reading a rational number and storing it.
2. Removing common factors and reducing a rational number to its smallest form.
3. Adding, multiplying and dividing rational numbers.
4. Printing a rational number.

Session 15

Write a program that maintains a telephone directory in a file. It should use one line for storing each record. Each record should consist of 2 fields, the name field and the telephone number field which are separated by a comma. The program should allow the user to add, delete or display data.

Session 16

Write a C program that administers an objective type test. The questions should be in one file and the correct answers should be in another file. The program should use a **struct** for storing the question and `fseek()` to move from question to question in the file. The program should keep a record of questions answered correctly. At the end of the test, it should print the marks as well as the questions that were wrongly answered.(You may find it useful to go through Program 10.12 in page 132 of Block 2).

Session 17

Create a linked list which holds the names of the cities New Delhi, Johannesburg, Rio de Janeiro, Copenhagen and Zurich in the same order, i.e. the head node must contain to New Delhi, the next node must contain Johannesburg and so on. Sort the list in lexicographic order by insertion sort using the funtion `strncmp()` for comparison. Afterwards, add the cities Madrid and London in appropriate places. Finally, the program should print the linked list.

Session 18(Compulsory)

Program 1

Write a program that reverses a string using a stack.

Program 2

Write a program that reads an expression from the terminal and checks if the brackets are properly matched.

Session 19(Compulsory)

Implement a queue containing integers as a linked list by implementing functions for creation of the queue, add an element of the queue, remove an element from the queue, to get the element at the top of the queue and the element at the bottom of the queue. Your linked list will have two pointers, one to the top and other to the bottom of the queue.

Session 20(Compulsory)

Create a binary search tree. Insert in the binary tree the words in the sentence ‘All the world is a stage, And all the men and women merely players.’ in the order they occur in the sentence. Then, do an in-order, pre-order and post-order traversals of the trees and list the keys in each node traversed.