Intro: ----------

Detailed Instructions on Project

Content: ----------

Expected features of various projects

* bc

unlimited precision real numbers.
operations: + - * / ^ << >> ^ % ; infix calculation with ()
variables
history
use of "."
sine, cosine, tangent() functions;
(Any other functionality specifically assigned to you)
Submission: In addition to code
one text file containing one expression (for which you have checked your code) per line.
Also Need to Submit: A file with tested commands - one per line. File named as test.txt

* Graphical Calculator

This one will be done using GTK or QT windowing toolkit libraries.
The interface should be like the gnome-calculator program.
unlimited precision real numbers.
Either Programming Mode or Scientific Mode to be implemented.

* Compression algorithms

Minimum two algorithms, as specified to you.
The program should be able to compress and decompress any file.
Program will run like this:
here the first two commands are for compression and
later two demonstrate decompression
./program -c1 <file> <compressed-file-name>
./program -c2 <file> <compressed-file-name>
./program -uc1 <compressed-file> <uncompressed-file>
./program -uc2 <compressed-file> <uncompressed-file>

* Encryption

The description is exacly like compression. Minimum two algorithms will be specified to you.

* Calendar

See google calender for an example.
A calendar application using GTK or QT or SDL.
One should be able to see infinite calendar.
Views: Daywise, weekwise, monthly, etc.
Functionality: creating events(repeated, non-repeated, etc), showing calendar for any date-date!, reminders, saving of calendar, restoring of calendar in ICS format.

Submission: Should submit a calendar with some ICS files already in your git repo.

Chain reaction Game:

* M x M board. Configurable by user as difficulty levle.
* N human players to be supported
* N-1 human and 1 computer player to be supported.
* Save and resume a game. Game should be saved to file.

Chess

* All rules of chess to be implemented properly.
* 2 human player mode
* 1 human + 1 software/computer player mode
* Save and resume a game. Game should be saved to file.
* The quality will be determined by quality of the computer player.
* Nice and intuitive user interface required in case of terminal based game.

*** Quiz**

The quiz program, running on command line should have at least following features:
creation of a question bank
various types of questions: multiple choice (any no. of choices), multiple correct answers, numerical answer questions, match-the-pair;
Selecting questions from question bank for a quiz;
storing a quiz, restoring a quiz, storing a running quiz session, restoring a running quiz-session; exporting the entire question bank, importing the question bank;
categorizing questions in question bank.

*** File I/O Library**

Write an implementation of the given FILE functions using the functions open, read, write, lseek, etc.
List of functions: fopen, fclose, fread, fwrite, fgetpos, fsetpos, feof, ftell, fseek
Test-suite is required.
Guidelines:
Read the manual pages completely
Try the functions completely to undestand how they work
Read stdio.h and see definition of struct FILE
Design a data structure to implement the FILF type. Get it checked from Abhijit.
Code.

*** Heap Manager**

Write your own implementation of the malloc, free, realloc, calloc, etc. functions related to memory management.
Test-suite required.
Note: The memory allocation functions request a "big chunk" of memory from the OS. For every request by programs to allocate (malloc/calloc/realloc) memory, they take a small portion of this big-chunk, record the location and size, and return the address. When free() is called, they know the size of the chunk from their record. Writing these functions involve handling the problems of internal and external fragmentation (read

about them online). Also read slab allocator, budddy allocator, etc. For this you may use the brk() system call.

*** Library management**

Have a look at the Koha software. Write a library management software like this. Use a graphical interface library like GTK or QT. NOTE, STRONGLY: You need to learn the basics of library science.
Functionality expected: See the list of books, seach on author/title/publisher; add books, delete books, update books; each book to have multiple copies; Users; Issue; Requests; Pending Requests; Fine on late return; Export the data as XML and import as XML;

*** Logic Gate Simulator**

Write a software which allows one to create logic gates, combine them to form different circuits, apply different inputs and see the output. Gates: NOT, AND, NOT, NOR, XNOR, OR, XOR, NAND
Use GTK or QT library.
Store a diagram in a file. Read the diagram from a file.

*** Simulation of sorting algorithms**

Write graphical simulation of at least 4 sorting algorithms. Code should support automated and manual data generation/insertion and stepwise execution of sorting algorithms. You can use GTK or QT or Ncurses library.

*** Animation of stack, queue, list**

Show animation of different implementations of stack, queue and list data types. The code should allow a choice of different implementations of the data structures (array based - multiple techniques, linked-structures with multiple techniques, etc).

*** sort command**

The sort command sorts any user input or files. It can sort files of any size (e.g. 10G, 100GB, etc.). This requires you to write external sort.
Handle following options: -k -n -b -d -i -r -m -t -u
Also Need to Submit: A file with tested commands - one per line. File named as test.txt

* tree command
The tree command prints a directory tree vertically, so that it looks like a tree.
You need to learn printing techniques using printf (e.g. use of special characters in formatting string, etc.).
Your code needs to implement a n-way tree or a nicely done recursive code.
Options to be handled: -a -d -R --ignore-case -filelimit -ofilename -u -s -t -r -J
Difficult option: -p pattern
Also Need to Submit: A file with tested commands - one per linne. File named as test.txt

*** tar command**

The tar program creates a "tarball" from a given set of files and folders. The tarball is a dump of all the files and folders, and also (has to) include metadata about the files/folders.
Implment these options: -c -v -f -x --delete --append --update -A -t

Also Need to Submit: A file with tested commands - one per linne. File named as test.txt

*** grep**

Implement these options: -r -i -v -f -w -c -m -b -q -H -h -e.
Additional option (more difficult), expected to be done: handling regular expressions in grep.
Also Need to Submit: A file with tested commands - one per linne. File named as test.txt

*** diff**

Implement the patch command as well.
Following options: -y -c -r -t -w -b -i
Hint: Know the maximum common subsequence problem and solve it.
Also Need to Submit: A file with tested commands - one per linne. File named as test.txt

*** Sudoku**

Program should be able to (a) Generate a sudoku and offer to user for solving (b) Users to solve a sudoku (c) Save a partially solved Sudoku in a file (d) Resume a Sudoku from a saved file (e) Solve a user input Sudoku (user will input in a file, or using your UI) (f) Auto solve a partially solved Sudoku (g) Give Hints to users (h) Have configuration options on size of sudoku (m x m, upto 25) and also difficulty level (no. of slots already filled in (i) use a graphical interface using GTK or QT

*** File format converters**

E.g. JPG to BMP, ODT to PDF, etc.
The programs should be able to handle all files of the source type. They should also offer options as given by the "convert" utility on Linux. (The options depend on the file types).

*** Text Editor:**
The editor should have interface of nano or vi.

Features required: type a file, use tab, newline, save file, open a new file, open existing file, navigate (line up, line down, char left, char right, page up, page down), insert text (before, after in the middle, etc - anywhere), copy-paste, cut-paste, search, search and replace.

Notes: You may use the ncurses library or the termio library.

Important points About Each Project

* The project should be a software that can REALLY be used for some real life work.
* If you are implementing any existing Unix commands then the behaviour of your software should be exactly like the existing command.
* If you are implmementing a GUI based software, then the design of user interface is a crucial factor in your evaluation. The user interface should be easy, intuitive.
* Some of the projects require you to write a test-suite. Some require specifying all the tested inputs in a text file.

* Imp: You should work on the design of your project first. Write down all the ADTs that you will need. Write down all the function prototypes for the functions that you plan to write. Show it to Abhijit once (by around 20th Sep).

On Writing a test-suite

* Some projects (FILE * library, malloc and free library, etc.) require writing a test suite. It is mentioned against those projects which require a test-suite. Typically it is required for libraries.
* The test suite is a set of programs (written in C or Shell or C + Shell, etc). The tester runs only one program and that program in turn runs other programs. Each program tests one or more functionality of the software and reports success/failure.
* You are supposed to write an extensive test-suite for testin your code.
* You will also be judged on the quality of the test-suite.

Tips to those implementing unix commands

* Useful C library functions: getopt(for handling command line options), opendir(opening a directory), readdir(reading a directory), stat(), chmod(), chown(), mkdir(), tmpnam(), tmpfile()
* Pay special attention to design of a data structure for storing the data. For example n-way tree, stack, stack of strings, array of strings, etc.
* Pay special attention to design of how to combine various options together. Normally if you don't do this in the begining, your code will become very complex.

Projects Tasks and Submission Process

Given below is a detailed description of expectations from various projects, and the submission process:

Submission Process:

a) create your own account on http://gitlab.com or http://git.fosscommunity.in (henceforth called 'server')
b) create a *private (IMP, private!)* repository on the server. Give a good name to the repo
c) Create a README.md file in the repo describing your project
d) Add the user abhijit13 with permissions as 'master' to your project
d) Submit the git clone link (with the ".git" in the end) on moodle. E.g. the link looks like this: https://gitlab.com/abhijit13/taasika.git
e) create your own pair of ssh-keys usinng ssh-keygen. Submit your own public key on server to enable write access for yourself.
READ:
https://docs.gitlab.com/ee/gitlab-basics/create-your-ssh-keys.html

Intermediate and continuous evaluation:

* Learn git on your own. Minimum following commands need to be learnt:
  init, add, commit, push, pull, rm, branch, checkout
* Keep pushing your code to the server every 2-3 days.
  Abhijit will keep 'pull'ing your code and checking the 'master' branch.