Flow (idea flow tracker)

- Need to think of a simple name, for time being lets call it `flow`.
- A simple terminal ncurses based application to track research ideas and the possible options that we take in the way to achieve conclusion.
- The flowchart blocks should be able to link to the code location.
- This is not supposed to be a simple terminal flow chart application.
- Upon clicking or pressing enter the block should open the full description otherwise it should just display the summary.
- The plan is to store data in a self-defined blob format.
    - Alternatives would be to store the data in json/sql/plain text format which could be parsed by other GUI/web clients.
- This tool could be part of any code repo.
    - This tool can be initiated like git init.
- The code needs to be written with modularity and re-usability in mind.
    - Maybe this needs to be broken down into simpler projects.
    - Like the idea drawer which will parse the idea blob files and draw a graph could be a project in itself.
    - **This needs more thought.**
- When the user chooses to open the commit SHA then a new split vim window should be opened with that commit. Obviously this will only happen when the tool is launched within a git repo.

Flow chart representation

```
-----------------        -----------------
|  Basic idea   |------> |     Step-1    |
-----------------        -----------------
                                  |
                                  |
                                  |
                                  |      -----------------
                                  ------>|     Step-2    |-------> and so on...
                                         -----------------
```

Need to create a better flow diagram in xfig

- **Need to draw a state machine for the idea editor flow**.
    - Draw a state machine catering to all decided shortcut keys.

Idea struct

```
struct idea_node{
    struct idea_node* parent_node; /* NULL for head node */
    u8 level_of_node; /* Level in which the node is stored. */
    u8 node_id; /* Store 0 in the head and store the local child id in children nodes. */
    DATE_TYPE creation_date; /* To be printed in the idea block */
    char summary[MAX_SUMMARY_LENGTH]; /* It should not be more than 80 chars long */
    char description[MAX_DESCRIPTION_LENGTH]; /* max 400 chars long */
    u3 number_of_branchouts; /* max 8. */
    struct idea_node *branchouts[MAX_BRANCHOUTS]; /* list of branchout blocks, max 8 branchouts.*/
    int* idea_node_offsets; /* Store the offsets of each branchout idea node from current parent node */
    char code_path[MAX_PATH_LENGTH]; /* For the editor to jump to, max 300 chars long. */
    COMMIT_SHA sha_id; /* Used to store the 20 byte commit sha used to view the commit*/
    struct links node_links[MAX_LINKS]; /* Store the relevant web links in this list */
}

struct idea_header{
    u8 global_idea_id; /* Store the global idea id */
    u8 total_nodes_in_idea; /* Store the total number of nodes in this idea flow. */
    u8 total_number_of levels; /* Store the total number of levels in this idea flow. */
    u8 nodes_in_level[MAX_NUMBER_OF_LEVELS]; /* Number of nodes at each level in this idea flow. */
}

struct links{
    char link_address[200];
    char link_description[200];
}
```

Storage of user data

Design rules

- Easier approach and to keep it light weight would be to create custom binary objects like git blobs.
  - Define a format for the binary blob.
  - Wrap content in magic numbers,(Need more info on this)
- We need to write the idea first with a logic of following to the children node from there.

Format of the binary blob

- An overall format could be starting with the header of the idea and then follow the below defined blob format.
- Header format:
  - A magic number of 16 bits i.e. `0x1729` to indicate the start of the header.
  - A separator like '|'.
  - Store the global id of this idea flow as a u8.
  - A separator like '|'.
  - Store the total number of nodes in this idea flow as a u8.
  - A separator like '|'.
  - Store the total number of level in this idea flow as a u8.
  - A separator like '|'.
  - Store the number of nodes per level in this idea flow.
  - A separator like '|'.
  - A magic number of 16 bits i.e. `0x1729` to indicate the end of the header.
- Head node would be stored just like a generic node.
- Generic node storing format.
  - An 8 bit number to indicate the idea node id.
  - A separator like '|'.
  - An 80 chars i.e. 80 byte long summary containing ASCII chars.
  - A separator like '|'.
  - An 400 chars i.e. 400 byte long description containing ASCII chars.
  - A separator like '|'.
  - Next a 3 bit number to indicate the number of branch outs from the node.
  - A separator like '|'.
  - An offset value for each branch out node from this node.
    - How to order the idea nodes to make access straight-forward.
    - Do we want to store the branchouts immediately after the parent to make mentioning of offsets easier in the parent struct.
    - Check the design of elf format to solve this problem.
    - Need to check how to store graphical data in binary files.
- Store the ideas like we would store them in an array.
  - Store the head first i.e level 0 and immediately following it store its branchouts.
  - After the first level branchouts are laid out, then store level 2 branchouts.
  - Start laying out the level 2 branchouts starting from the children of the first node in level 1.
  - So on. . . .

Writing logic

Reading logic

Reading of head node

- The head node portion of the blob would be read first.
  - Read the 16-bit magic number.
  - Read the 8-bit number indicating the total number of nodes in this idea flow.

Reading of generic node

Data type specific rules

- To keep it easier let's fix the total number of nodes in a single idea to `256`.
  - Lets call it `MAX_NUMBER_OF_NODES_PER_IDEA`.
- To keep it simple let's fix the total number of levels in a single idea to `256`.
- Maybe we would need to fix the max number of nodes in a single level for memory constraints. Need to think about this.

Alternatives for storage of user data would be

- MySQL.(don't know how to store data which keeps on changing but most probably most efficient)
- Plain text files.(easier but not efficient)
- Json files.(easier)

custom ncurses interface

- Need to implement basic housekeeping functions like:-

  ```
  int draw_idea(struct idea* current_idea);
  int save_idea(struct idea* current_idea);
  int delete_idea(struct idea* current_idea);
  int create_a_copy_of_idea(struct idea* current_idea);
  ```

- Need to implement flow chart editing functions like:-

  ```
  int add_idea_block_to_the_right_of(struct idea_node* current_node);
  int add_idea_block_to_the_top_of(struct idea_node* current_node);
  int add_idea_block_to_the_bottom_of(struct idea_node* current_node);
  int copy_this_block(struct idea_node* copy_source);
  int paste_to_this_block(struct idea_node* paste_desitination);
  int edit_this_block(struct idea_nore* current_node);
  ```

- Need to implement an idea window which will be a simple ncurses screen with keyboard enable navigation.
- Should be able to parse any given idea and draw a simple flowchart.
- No low level routines should be exposed to the main application.
- Press `c` for copy, `e` for edit, `v` for paste, and `q` for exit.
- Edit will open the idea node in an edit window which will show all parameters in a vim window and ready for edit.
- Whenever an idea block is added the automatic creation date should be printed and stored.
- **Need to draw a state machine for this**.

Dir structure of `.flow` dir

- `.flowconfig` : config file.
  - Would need to write a small parser for parsing it.
  - Use `yacc` and `bison` for it.
  - It would be a good exercise to write a small parser in C.
  - Format of the configs would be simply `CONFIG_OPTION=<value>`.
  - Need to document all the config options in man page.
  - The default config file would have all options enabled if option is boolean otherwise a default value would be provided.
- Can store the `ideas` dir inside the `.flow` dir,
  - All `.idea` files would be stored inside it.
  - These `.idea` files would be binary blobs with self-defined format.

Control flow of the application

- If launched without options then usage and help is printed.
- If launched with the option `idea_tracker` then the user is presented with home screen.
- Need to make a list of the required options. Will document it in the man page.

Home screen

- Welcome message is printed.
- If launched with proper options then it will look for a dir named `.flow` for config options.
- If `.flow` dir is not found then it will create it.
- It will ask for 2 options i.e. what is the purpose research or code.
- Lets say research is selected.
- The control is transferred to the idea listing segment.

Idea listing

- Then it will look for a dir named `ideas` in the `.flow` dir and if it is not present then it will ask to create it.
- If the `ideas` dir is already present then it will display the one line summary of all the ideas as a bullet list.
- Next it will give an option to select an idea number through a keyboard navigated selection.
- Once an idea is selected the program will pass on the control of idea files to the idea parser.

Idea parser

- It will check and parse the idea blobs and if try to pass on the information to the idea drawer screen.
- If no idea nodes are found then it will pass and empty parsed blob to initiate the drawing of the first block.

Idea drawer screen

- The idea drawer screen will try to draw the idea by using the parsed information.
- If empty blob received then it will draw the first block of the idea and pass on the control to the user to fill it.
- The user can save the idea here once filled and exit the drawer screen.
- The control will now be passed on to the idea listing segment.

Creating new idea

- By pressing `o` the user would be able to create a new idea. For simplicity I will keep the ideas sorted by their ids so the new idea would always be created at the end of the list.

Returning to last working idea

- An optional option when the application would be relaunched be to directly to go to the last idea the user was working on.

Documentation generation

API doc generation from code comments

- https://github.com/doxygen/doxygen

Man page

- Look into `scdoc` to generate man pages easily.

Makefile Refs

- https://makefiletutorial.com

Library Refs

- https://en.cppreference.com/w/c/links/libs
- https://github.com/oz123/awesome-c