# About

TBB is a tool which allows users to browse image boards from the terminal. Different levels such as boards, catalogs, and threads are navigated with user inputted commands and the output is displayed in color coded text in the terminal.

# Structure

The main method instantiates an instance of the interface class. This interface contains an instance of Client to request/retrieve information and an instance of PrintStream in order to display content to the user. Before being printed content is formatted by the different classes in the ImageBoard folder based on its type.

# Requirements

* g++ version 4.8 or greater
* Curl
* Jsoncpp
* readline

# Class Specifications

## Cache Manager

Private members:

* CacheManagerState state
  + Holds the current state of the cache manager / Determines how the cache should be refreshed (see enum CacheManagerState)
* CURLCode curl
  + CURLCode representing the status of request
* String cache\_dir
  + Path of the directory in which cache files are stored
* String config\_file
  + Path of config file
* String cache\_log
  + Name of the cache file within the cache directory
* fstream log
  + Stream used to write cache files
* bool cacheFile(const string&, const string&), createDir(const string&);
  + Creates a path with nested directories. If any directory in the path already exists, moves along the path until finds one which doesn't
* inline bool dirExists(const string&);
  + Returns a bool corresponding to if the directory specified by the inputted string exists
* string getURL(const Target&);
  + Creates and returns the URL to make a request to, corresponding with the inputted target

Public members:

* void setState(const CacheManagerState);
  + Sets state to the inputted state
* string getPath(const Target&);
  + Returns the file path to the given target
* bool prepCache(const Target&);
  + Prepares a target to get cached. Returns whether or not the target is in the cache.

Constructors:

* CacheManager();
  + sets state to REFRESH, cache\_dir to ~/.tbb/cache, log file to ~/.tbb/cache/.cache.log. If the file or directory does not exist the function creates them.

Enums:

* enum CacheManagerState {

NEVER\_REFRESH = 0,

FORCE\_REFRESH = 1,

REFRESH = 2

};

## Client

Private Members:

* ​​CacheManager CM;
  + The currently used cache manager
* Target target;
  + The current target
* Json::Value getContent();
  + Read the content of the current target from cache into a Json::Value

Public Members:

* TargetType getCurrentTarget();
  + Returns current target type
* string getCurrentPage();
  + returns header of page that contains info on the current view and target
* string getPage();
  + returns a string representation of the current page
* void goToBoardList();
  + Sets target type to BOARD\_LIST
* void goToBoard(string);
  + Takes in a string (board), sets target type to CATALOG, and sets the target board to the inputted board
* void goToCatalog(string);
  + Takes in a string board, sets target type to CATALOG, and sets the target board to the inputted board
* void goToThread(unsigned int);
  + Takes in a int representing thread number, sets target type to THREAD, and sets the target thread to the inputted thread

Constructors

* Initializes target type to BOARD\_LIST

## Catalog

Private Members:

* Page \*pages;
  + Array of all the pages in the catalog
* size\_t page\_count;
  + Amount of pages in catalog
* Json::Value info;
  + Content of the catalog represented in JSON

Public Members:

* friend ostream& operator << (ostream&, const Catalog&);
  + Creates and returns output stream for displaying Catalog
* string toString();
  + returns string representation of catalog

Constructors:

* Catalog();
  + initializes pages to null
* Catalog(const Json::Value&);
  + takes in a JSON representation of the content, initializes page\_count to the size of the inputted content, and populate the pages array with the corresponding page data

## Post

Protected Members:

* Json::Value info;
  + Content of the Post represented in JSON

Public Members:

* friend ostream& operator << (ostream&, const Post&);
  + Creates and returns output stream for displaying Post
* string toString();
  + returns post in string format

Constructors:

* Post();
  + Creates empty post object
* Post(const Json::Value&);
  + Initializes info to the inputted JSON

## Original Post

Inherits from Post

Public Members:

* friend ostream& operator << (ostream&, const OriginalPost&);
  + Creates and returns output stream for displaying OriginalPost
* string toString();
  + Returns original post in string format

Constructors:

* OriginalPost();
  + Creates OriginalPost object with default Post constructor
* OriginalPost(const Json::Value&);
  + Initializes Post object with inputted JSON

## Reply

Inherits from Post

Constructors:

* Reply();
  + Initializes reply with a default Post object
* Reply(const Json::Value&);
  + Initializes reply as a Post with the inputted JSON

## Thread

Private Members:

* OriginalPost OP;
  + The original post of the given thread
* Reply \*replies;
  + Array of replies to the given thread
* size\_t num\_replies;
  + size of replies array

Public Members:

* friend ostream& operator<< (ostream&, const Thread&);
  + Creates and returns output stream for displaying Thread
* string toString();
  + returns post in string format
* OriginalPost getOP();
  + Returns the original post of thread
* Reply \*getReplies(size\_t &);
  + returns an array of all replies for Thread

Constructors:

* Thread();
  + Creates empty thread object with num\_replies set to 0
* Thread(const Json::Value&);
  + Initializes reply\_count to the amount of replies in the inputted thread data and popullates the replies array with the corresponding reply data

## Interface

Private Members:

* const char \*prompt;
  + Input command given by the user
* Client client;
  + Client object which the interface navigates
* PrintStream out;
  + The printstream which displays the interface
* void parseCommand(char \*, unsigned char&);
  + Takes in user command in form of a char array and reference to flags. Parses command, performs command action, and sets appropriate flags

Public Members:

* void loop();
  + Loops through cycle of getting user input and updating interface until program is quit

Constructors:

* Interface();
  + Initializes out to cout and prompt to its initial state

## PrintBuff

Heavily Sampled from: <https://stackoverflow.com/questions/9599807/how-to-add-indention-to-the-stream-operator>

Inherits from streambuf

Public Members:

* void set\_indent(int);
  + Sets indent size to the inputted int
* size\_t indent\_width, width, def\_width, count, tab\_count;
* static const int tab\_width = 8;
  + Defines size of a tab
* string prefix;
  + The string that is placed before the content
* streambuf\* sbuf;
* string buffer;

Constructors:

* PrintBuf(int,std::streambuf\*);

## PrintStream

Heavily Sampled from: <https://stackoverflow.com/questions/9599807/how-to-add-indention-to-the-stream-operator>

Inherits from ostream

Private Members:

* static int getWindowSize();
  + returns window size
* static int window\_size;

Public Members:

* PrintStream& indent(int);
  + Sets the indent size to the inputted int by calling PrintBuf.set\_indent()
* PrintBuf& getPrintBuf();
  + Returns PrintBuf Object

Constructors:

* PrintStream(size\_t, std::ostream&);
  + Creates a PrintBuf Object with the inputted width and ostream
* PrintStream(std::ostream&);
  + Creates a PrintBuf Object with width of the current window and the inputted ostream

## HtmlToPlain

Public Members:

* string htmlToPlain(string);
  + Takes in string of html and returns the corresponding plain text string

## TBB

Contains main method. Creates instance of Interface and loops it.

## Colors.h

ANSI escape codes for all necessary colors

## Target.h

Enums:

* enum TargetType {

BOARD\_LIST = 1,

THREAD\_LIST = 2,

CATALOG = 3,

THREAD = 4,

MEDIA = 5

};

# Challenges

Our largest challenge was setting attainable goals and sticking to them. In the early stages of our project we cycled through several ideas on how the input process, display, and cache should work. Simply because there were so many things we wanted to do, we stretched ourselves a bit too thin and did not focus on what needed to be done in the moment. Once we settled on how to do these things we only ran into minor problems and bugs.

# Conclusion

We achieved our initial goal of building a tool that allows us to navigate image boards in the terminal. Although we did not have the time to add every feature we wanted, our code is structured modularly enough for easy implementation of support for other image boards and social media in the future. This robust modularity became one of our goals after the uncertainty mentioned in the challenges section and in the end we successfully achieved it. Overall, we are satisfied with how tbb functions and have found it useful for image board browsing in our daily lives.