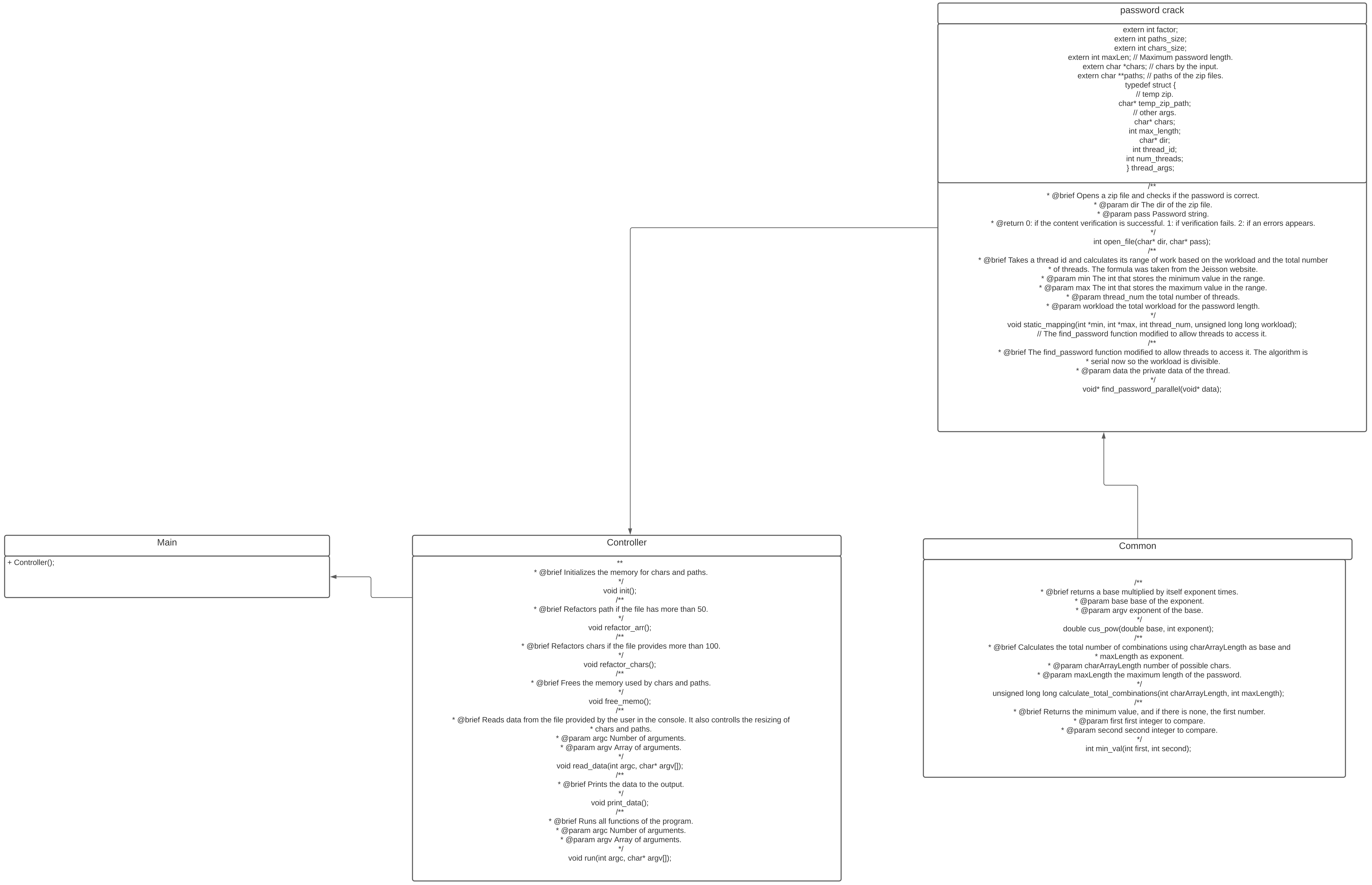

```
extern int factor;
extern int paths_size;
extern int chars_size;
extern int maxLen; // Maximum password length.
extern char *chars; // chars by the input.
extern char **paths; // paths of the zip files.
typedef struct {
// temp zip.
char* temp_zip_path;
// other args.
char* chars;
int max_length;
char* dir;
int thread_id;
int num_threads;
} thread_args;
```

```
/**
* @brief Opens a zip file and checks if the password is correct.
* @param dir The dir of the zip file.
* @param pass Password string.
* @return 0: if the content verification is successful. 1: if verification fails. 2: if an errors appears.
*/
int open_file(char* dir, char* pass);
/**
* @brief Takes a thread id and calculates its range of work based on the workload and the total number
* of threads. The formula was taken from the Jeisson website.
* @param min The int that stores the minimum value in the range.
* @param max The int that stores the maximum value in the range.
* @param thread_num the total number of threads.
* @param workload the total workload for the password length.
*/
void static_mapping(int *min, int *max, int thread_num, unsigned long long workload);
// The find_password function modified to allow threads to access it.
/**
* @brief The find_password function modified to allow threads to access it. The algorithm is
* serial now so the workload is divisible.
* @param data the private data of the thread.
*/
void* find_password_parallel(void* data);
```

```
+ Controller();
```

```
**
* @brief Initializes the memory for chars and paths.
*/
void init();
/**
* @brief Refactors path if the file has more than 50.
*/
void refactor_arr();
/**
* @brief Refactors chars if the file provides more than 100.
*/
void refactor_chars();
/**
* @brief Frees the memory used by chars and paths.
*/
void free_memo();
/**
* @brief Reads data from the file provided by the user in the console. It also controlls the resizing of
* chars and paths.
* @param argc Number of arguments.
* @param argv Array of arguments.
*/
void read_data(int argc, char* argv[]);
/**
* @brief Prints the data to the output.
*/
void print_data();
/**
* @brief Runs all functions of the program.
* @param argc Number of arguments.
* @param argv Array of arguments.
*/
void run(int argc, char* argv[]);
```

```
/**
* @brief returns a base multiplied by itself exponent times.
* @param base base of the exponent.
* @param argv exponent of the base.
*/
double cus_pow(double base, int exponent);
/**
* @brief Calculates the total number of combinations using charArrayLength as base and
* maxLength as exponent.
* @param charArrayLength number of possible chars.
* @param maxLength the maximum length of the password.
*/
unsigned long long calculate_total_combinations(int charArrayLength, int maxLength);
/**
* @brief Returns the minimum value, and if there is none, the first number.
* @param first first integer to compare.
* @param second second integer to compare.
*/
int min_val(int first, int second);
```