TOPIC: Multi-platform Development

Objective of the assignment:

- Recap:
    - Working with files
    - Dynamic memory allocation
- Using pointers

Statement

Implement a hash table in C that will store words. The operations to be implemented for the hash table are as follows:

Operation	Description
add <word>	Adds the word to the hash table (duplicates are not allowed)
remove <word>	Removes the word from the hash table (it is not mandatory for the word to exist)
find <word> [<output_file>]	Searches for the word in the hash table → write **True** or **False** on a new line in the specified file, or to the console if this parameter is missing.
clear	Clears the hash table.
print_bucket <bucket_index> [<output_file>]	Writes the words in the specified bucket, on a single line and separated by spaces in the specified file, or to the console if this parameter is missing; bucket_index is valid.
print [<output_file>]	Prints all buckets on different lines, starting from bucket 0, in the specified file, or to the console if this parameter is missing.
resize double	Doubles the size of the hash table (buckets will be iterated in order and words will be redistributed).
resize halve	Halves the size of the hash table (buckets will be iterated in order and words will be redistributed; surplus memory will be deallocated).

These commands will appear one per line.
The program will receive a series of parameters:

- The first parameter is the initial size of the hash table.
- The following parameters are optional and represent a list of input files from which data will be read. If they are missing, the input will be read from STDIN. **Attention**: if multiple input files are specified, all operations apply **to the same hash table**, in the order they were provided in the command line. If a file does not exist, it will be ignored.

Usage example:
./executable SIZE [files]

./assigment1 256 hash1.in hash2.in
where **hash1.in** contains:

> **add** tema
> **add** hash
> **print** hash.out
> **find** tema hash.out
> **remove** tema
> **find** tema hash.out
> **print** hash.out
> **resize double**
> **print**
> **print_bucket** 185 hash2.out

The implemented hash table will contain **SIZE** buckets. Each bucket will store the words in the order they were inserted. For the resize operation, the buckets will be iterated in order and redistributed. The words in a bucket will be iterated starting from the oldest to the most recent.

General notes:

- Values inserted into the hash table are words matching the regex [A-Za-z]+.
- An array cannot contain duplicates.
- There are no limitations on the length of a bucket.
- Insertion into an array (bucket) is done at the end of it.
- The hash function to be used (for the entire assignment) is defined in hash.c. No other function can be used. The function archive is available in the resources below.
- The program must execute commands in the order they were received or read from the file(s).
- Empty lines in the input file should be ignored (the program does nothing and moves to the next line).
- Writing to files must be done in append mode.
- If the hash table size is odd (2k+1), after halving, its size will be k.
- Both the hash table size and word length will be represented as unsigned 32-bit numbers.
- The empty string is not valid.
- The hash table size will always be positive.
- The generated executable will be named tema1 on Linux and tema1.exe on Windows.
- The maximum length of a command (operation and associated word) is 20,000 characters.
- The buffer used to read commands can be declared with a static size.
- The desired behavior for resize is as follows: a new hash is created, and the buckets in the old hash are iterated and added to the new hash.
- The hash table CANNOT be implemented using statically allocated arrays.
- If a bucket is empty, NO empty line should be inserted.
- Any error (including receiving an incorrect command from a file or stdin) must be reported to stderr with a relevant message. You can use the DIE macro. The application must return a value less than 0 in main() if errors were encountered during execution.