

# LABWORK 4: Word Count

**Student's name: Hoang Quoc Minh Quan**

**Student's ID: BI12-363**

## I. Choice of Mapreduce implementation

The specific MapReduce implementation in my C code was chosen for its simplicity and effectiveness in performing a word count on a text file. Here are the key reasons for choosing this implementation:

- **Straightforward Implementation:** The implementation is relatively simple and easy to understand, making it suitable for educational purposes and initial exploration of map-reduce concepts.
- **Clear Separation of Concerns:** The code clearly separates the mapping (map function) and reducing (reduce function) phases, which aligns well with the fundamental map-reduce paradigm.
- **Minimalistic Design:** The implementation uses basic C programming constructs without dependencies on external libraries, making it portable and lightweight.
- **Illustrative Purposes:** This implementation serves as a practical example of how map-reduce can be applied to a simple problem, making it valuable for learning and understanding the core principles of map-reduce.

## II. Workflow

### *a. Mapper*

The “map” function in this implementation processes each line of input text to tokenize words and populate a “KeyValuePair” array (“pairs”) with word counts.

1. **Input:** The “map” function takes a “line” of text as input.

```
// Mapper function
void map(char *line, KeyValuePair *pairs, int *pair_count)
```

2. **Tokenization:** It iterates through each character in line.
  - If the character is alphanumeric (“isalnum(\*line)”), it appends the lowercase version of the character to a “word” buffer.

```

20         if (isalnum(*line)) {
21             word[i++] = tolower(*line);

```

- When a non-alphanumeric character is encountered, the accumulated “word” (if not empty) is added to the “pairs” array with an initial count of 1.

```

22     } else if (i > 0) {
23         word[i] = '\0';
24         // Add key-value pair to the array
25         strcpy(pairs[*pair_count].key, word);
26         pairs[*pair_count].value = 1;

```

3. **Increment Pair Count:** The “pair\_count” pointer is incremented for each new “KeyValuePair” added to the “pairs” array.

## b. Reducer

The “reduce” function aggregates word counts from the “pairs” array to produce the final word count results.

1. **Input:** The “reduce” function takes the populated “pairs” array and “pair\_count” (number of items in the array) as input.

```

void reduce(KeyValuePair *pairs, int pair_count) {

```

2. **Aggregation:**

- It iterates through each “KeyValuePair” in the “pairs” array.

```

for (i = 0; i < pair_count; i++) {

```

- For each unique word (key), it sums up the counts (value) of all occurrences by iterating over the remaining elements of the array.

```

    for (j = i + 1; j < pair_count; j++) {
        if (strcmp(pairs[i].key, pairs[j].key) == 0) {
            total_count += pairs[j].value;
            pairs[j].value = 0; // Mark as visited
        }
    }
}

```

- Words that have been aggregated are marked as visited “(pairs[j].value = 0)” to avoid redundant processing.

3. **Output:** For each unique word, it prints the word and its total count (“total\_count”) to produce the final word count results.

```
// Print the word and its count
if (pairs[i].value != 0) {
    printf("%s\t%d\n", pairs[i].key, total_count);
}
```

**c. Figure**

Here’s is the figure that illustrated the workflow of Mapper and Reducer:

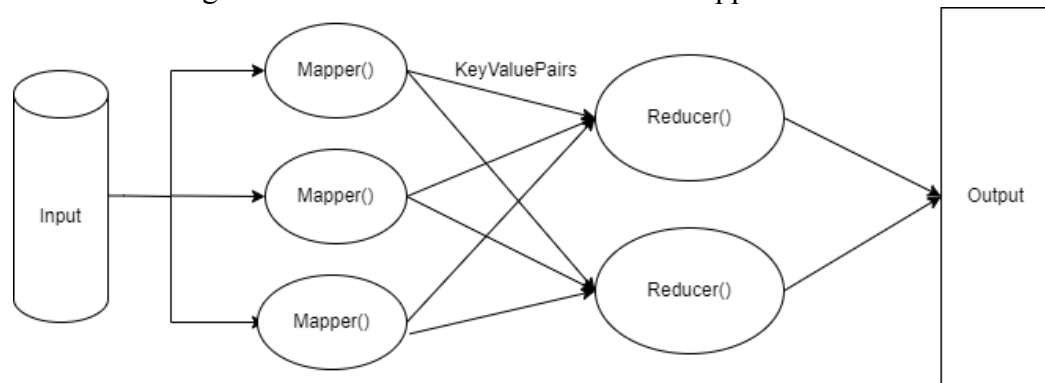
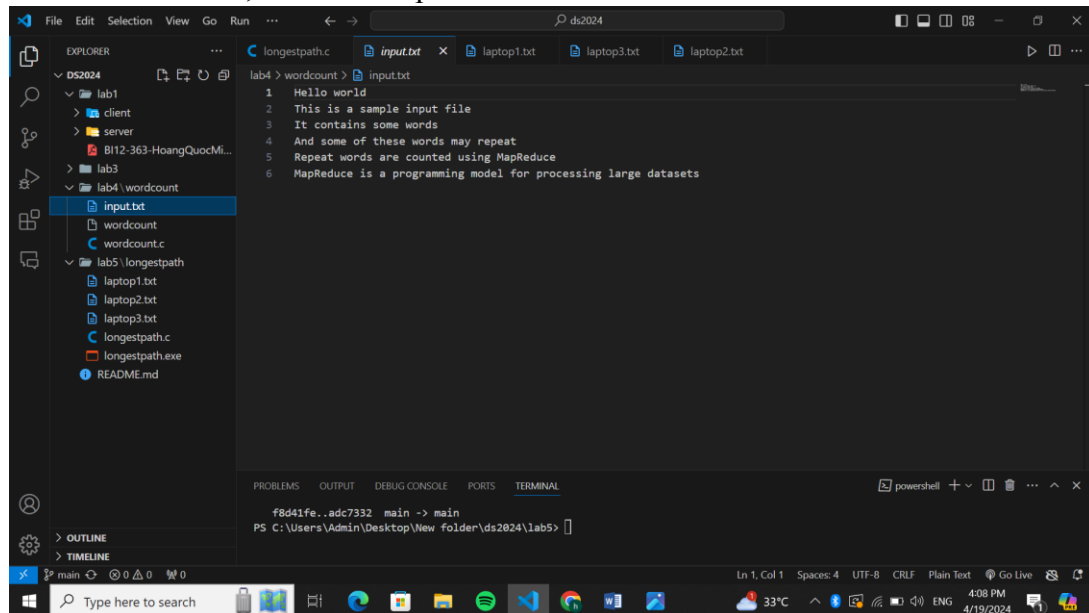


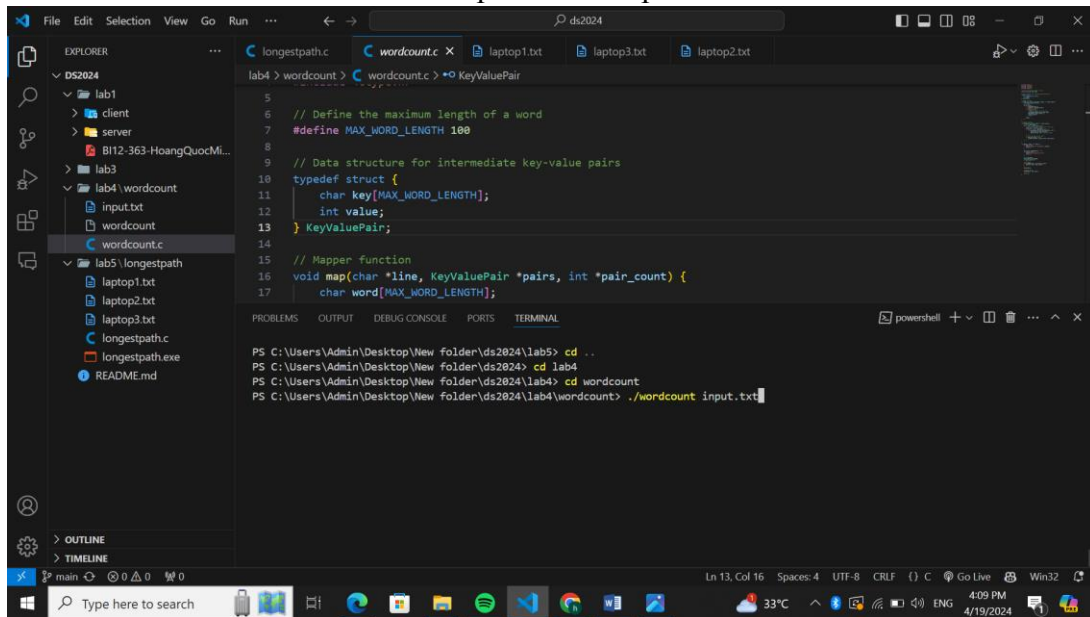
Figure 1. Workflow of MapReduce

### III. Implementation

This is the test file, which is “input.txt”



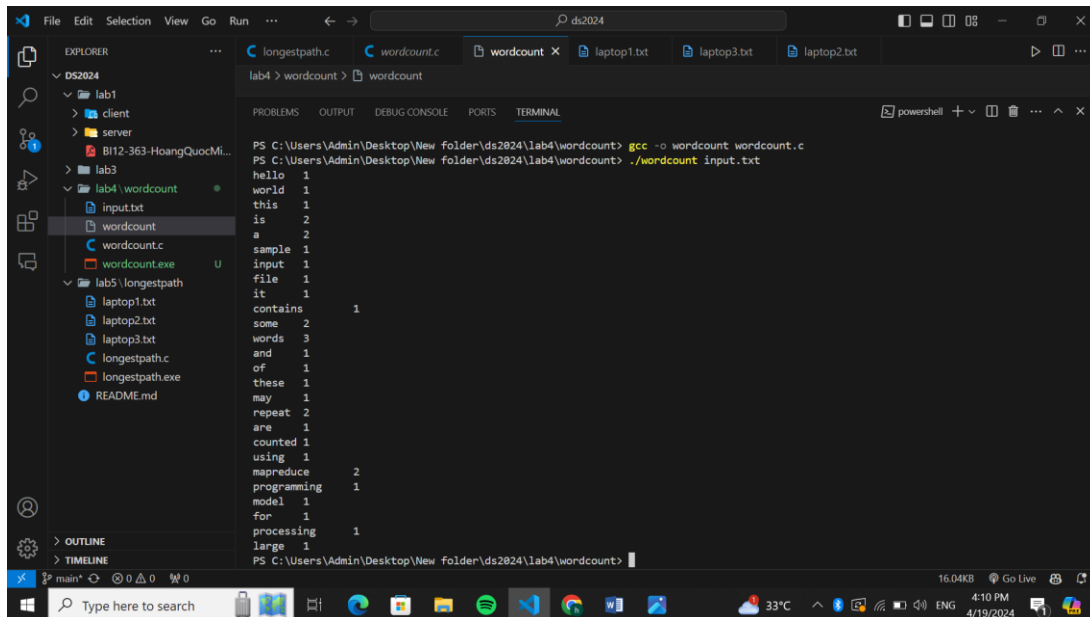
Now we run the wordcount.c with input is the “input.txt”



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows a project structure with folders 'lab1', 'lab3', 'lab4', and 'lab5'. The 'lab4' folder contains 'wordcount' and 'wordcount.c'. The 'wordcount.c' file is open in the editor. The code defines a structure for key-value pairs and a map function. The terminal window shows the following commands and output:

```
PS C:\Users\Admin\Desktop\New folder\ds2024\lab5> cd ..
PS C:\Users\Admin\Desktop\New folder\ds2024> cd lab4
PS C:\Users\Admin\Desktop\New folder\ds2024\lab4> cd wordcount
PS C:\Users\Admin\Desktop\New folder\ds2024\lab4\wordcount> ./wordcount input.txt
```

And this is the result



The screenshot shows the Visual Studio Code editor with the file explorer on the left. The file explorer shows the same project structure as the previous screenshot. The 'wordcount' file is open in the editor. The terminal window shows the following commands and output:

```
PS C:\Users\Admin\Desktop\New folder\ds2024\lab4\wordcount> gcc -o wordcount wordcount.c
PS C:\Users\Admin\Desktop\New folder\ds2024\lab4\wordcount> ./wordcount input.txt
```

The output of the program is as follows:

```
hello 1
world 1
this 1
is 2
a 2
sample 1
input 1
file 1
it 1
contains 1
some 2
words 3
and 1
of 1
these 1
may 1
repeat 2
are 1
counted 1
using 1
mapreduce 2
programming 1
model 1
for 1
processing 1
large 1
```