ENGG1110 Problem Solving by Programming (2018-2019 Term 1)
Project – Word Search Puzzle

---

1. Introduction

A Word Search Puzzle is a classic game to find words hidden in a **letter grid**, where all the words are provided in a **word list**.

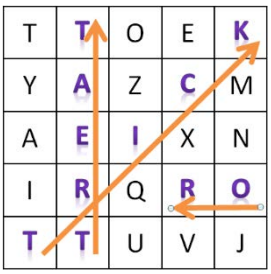
The letter grid consists of some cells organized in a square or a rectangle shape, where each cell contains a letter. An example is shown as follows:

T	T	O	E	K
Y	A	Z	C	M
A	E	I	X	N
I	R	Q	R	O
T	T	U	V	J

The word list contains some words to be found in the letter grid. An example is shown as follows:

TRICK
OR
TREAT

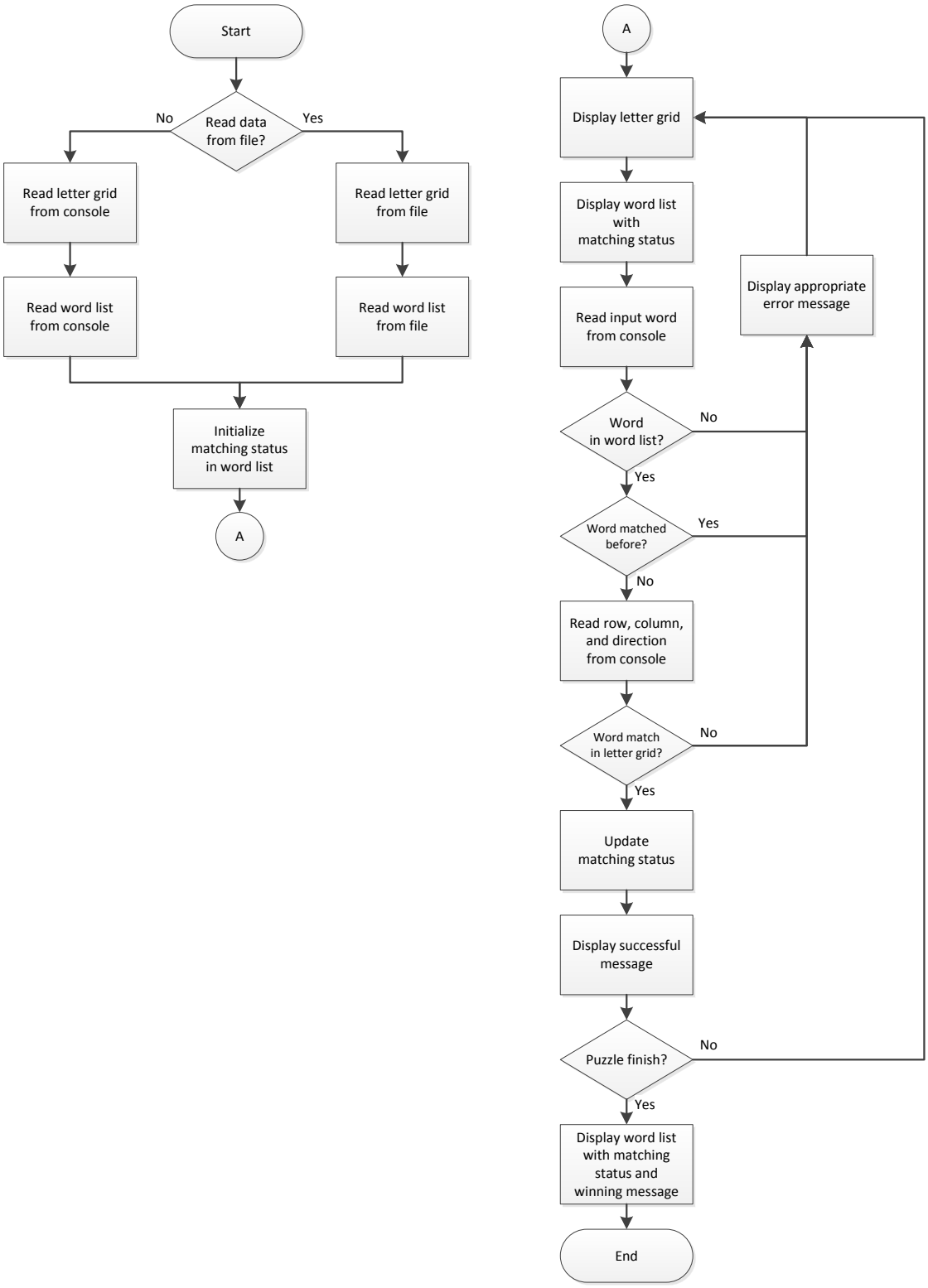
The words in the word list can be found in one of the 8 directions in the letter grid, i.e., 2 for horizontal (i.e., left to right or right to left), 2 for vertical, and 4 for diagonal. An example is shown as follows:



In this project, your task is to use **Code::Blocks** in **Windows** to create a Console Application of a Word Search Puzzle using C language.

2. Program Design

A skeleton code (in **SkeletonCode.c**) is given and you are required to complete it using **Code::Blocks** in this project. The program flow is shown as follows:



3. Schedule

The following shows the suggested schedule, the code submission deadline, and the date for the project demo and presentation:

Week	Date	Tasks
9	Oct 31	• Choose Data Source: Console or File? (Section 4.2) • Read Data from Console (Section 4.3) • Initialize Matching Status (Section 4.5) • Display Letter Grid and Word List (Section 4.6) By the end of this week, your program should be able to: ○ Print the letter grid and the word list that are read from the console
10	Nov 7	• Read Data from File (Section 4.4) • Read Input Word from Console and Check Word List (Section 4.7) By the end of this week, your program should be able to: ○ Print the letter grid and the word list that are read from the file ○ Read an input word from the user and decide whether it is in the word list or matched before
11	Nov 14	• Read Row, Column, and Direction from Console (Section 4.8) • Check Letter Grid (Section 4.9) • Check Winning Condition (Section 4.10) By the end of this week, your program should be able to: ○ Allow a player to finish the whole puzzle
12	Nov 21	• Print Secret Table (Section 5) • Final Testing on Lab Computer By the end of this week, your program should be able to: ○ Allow a player to enter the secret code to print the secret table ○ Run smoothly on a lab computer, which will be used in the project demo and presentation
13	**Nov 26**	• Project Submission (by 23:59 on Blackboard)
	Nov 28	• Project Demo and Presentation (in the lesson)

4. Detailed Design

4.1 Macros and Variable Declarations

Suppose the macro **MAX_GRID** is defined in the beginning of the source code:

```
#define MAX_GRID 10
```

And we have the following in the other part of the source code:

```
char letterGrid[MAX_GRID][MAX_GRID]
```

The above will be replaced by

```
char letterGrid[10][10]
```

automatically before the source code compiles.

The following table shows the macros defined in the beginning of the skeleton code. They are mainly used for the declarations of arrays and parameters.

Macro	Value	Explanation
MAX_GRID	10	Maximum size of the letter grid, which is a square The value is no larger than 10
MAX_WORDLIST	8	Maximum size of the word list
MAX_WORD	11	Maximum number of bytes of a word in the word list and a word inputted by the user, including the terminating NULL character The value is no larger than MAX_GRID + 1
MAX_FILENAME	260	Maximum number of bytes in a filename, including the terminating NULL character

Your program should run correctly even if the macro values are changed reasonably.

Besides, you cannot use any global variables in this project. In other words, all variables must be declared inside functions.

4.2 Choose Data Source: Console or File?

The program will first ask whether to read the letter grid and the word list from files with the following prompt:

```
Read data from file [Y/N]?
```

You can assume that the user must input either Y or N. You are required to implement this feature in the **main** function.

4.3 Read Data from Console

If the user did not choose to read the data from file, the program will simply read the letter grid and the word list from the console. The following two functions have already been implemented in the skeleton code. You are required to study them first and then invoke them from the **main** function:

- **`int readLetterGridFromConsole(char letterGrid[MAX_GRID][MAX_GRID])`**
  - Read the letter grid from console into the 2D array parameter **letterGird**
  - Return the size of the letter grid
- **`int readWordListFromConsole(char wordList[MAX_WORDLIST][MAX_WORD])`**
  - Read the word list from console into the 2D array parameter **wordList**
  - Return the size of the word list

The program will first ask the user for the size of the letter grid with the following prompt:

```
Enter the size of the letter grid:
```

You can assume that the input must be an integer of at least 2 but no larger than **MAX_GRID**. Let the input be **gridSize**. Then, the program will ask the user for the letter grid with the following prompt:

```
Enter the letter grid:
```

You can assume that the input must be **gridSize** lines of uppercase letters, where the number of letters on each line is **gridSize**. An example is shown as follows (underlined characters in blue are user inputs):

```
Enter the size of the letter grid:
5
Enter the letter grid:
CTOEK
YAZCM
AEIXN
IRQRO
TTUVJ
```

You can assume that:

1. **gridSize** is at least 2 but no larger than **MAX_GRID**.
2. All letters are uppercase letters.

After reading the letter grid, the program will ask the user for the size of the word list with the following prompt:

```
Enter the size of the word list:
```

You can assume that the input must be an integer of at least 1 but no larger than **MAX_WORDLIST**. Let the input be **listSize**. Then, the program will ask the user for the word list with the following prompt:

```
Enter the word list:
```

You can assume that:

1. **listSize** is at least 1 but no larger than **MAX_WORDLIST**.
2. The length of each word is at least 2 but no longer than **MAX_WORD – 1**.
3. There are no duplicate words in the word list.
4. Each word must appear once and only once in the letter grid in one of the 8 directions.
5. All letters are uppercase letters.

An example is shown as follows (underlined characters in blue are user inputs):

```
Enter the size of the word list:
3
Enter the word list:
TRICK
OR
TREAT
```

After reading the word list, we have **listSize** = 3 and

wordList[0] is “TRICK”
wordList[1] is “OR”
wordList[2] is “TREAT”

4.4 Read Data from File

If the user chose to read the data from file, the program will ask for two filenames and then read the letter grid and the word list from the specified files. You are required to complete and then invoke the following two functions from the **main** function:

- **`int readLetterGridFromFile(char letterGrid[MAX_GRID][MAX_GRID])`**
  - Read the letter grid from file into the 2D array parameter **letterGird**
  - Return
    - the size of the letter grid if successful
    - -1 if there is any file reading error
- **`int readWordListFromFile(char wordList[MAX_WORDLIST][MAX_WORD])`**
  - Read the word list from file into the 2D array parameter **wordList**
  - Return
    - the size of the word list if successful
    - -1 if there is any file reading error

The program will first ask the user for the filename of the letter grid with the following prompt:

```
Enter filename of the letter grid:
```

The file format is described as follows. The first number indicates the size of the grid, denoted as **gridSize**. Then, there are **gridSize** lines of uppercase letters, where the number of letters on each line is **gridSize**. An example is shown as follows:

LetterGrid.txt

```
5
CTOEK
YAZCM
AEIXN
IRQRO
TTUVJ
```

You can assume that:

1. The length of the filename input by the user is no longer than **MAX_FILENAME – 1**.
2. The file format must be correct.
3. **gridSize** is at least 2 but no larger than **MAX_GRID**.
4. All letters are uppercase letters.

If the file cannot be found or there is any IO error, the program will display the following message and terminate.

```
Error in reading the letter grid file. Program terminates.
```

After reading the letter grid from the file, the program will ask the user for the filename of the word list with the following prompt:

```
Enter filename of the word list:
```

The file format is described as follows. The first number indicates the size of the list, denoted as **listSize**. Then, there are **listSize** words on each line. An example is shown as follows:

WordList.txt
3
TRICK
OR
TREAT

You can assume that:

1. The length of the filename input by the user is no longer than **MAX_FILENAME – 1**.
2. The file format must be correct.
3. **listSize** is at least 1 but no larger than **MAX_WORDLIST**.
4. The length of each word is at least 2 but no longer than **MAX_WORD – 1**.
5. There are no duplicate words in the word list.
6. Each word must appear once and only once in the letter grid in one of the 8 directions.
7. All letters are uppercase letters.

After reading the word list, we have **listSize** = 3 and

wordList[0] is “TRICK”
wordList[1] is “OR”
wordList[2] is “TREAT”

If the file cannot be found or there is any IO error, the program will display the following message and terminate.

```
Error in reading the word list file. Program terminates.
```

4.5 Initialize Matching Status

After reading the letter grid and the word list from the console or the files, you are required to initialize the matching status of each word in the word list in the **main** function. In the skeleton code, the matching status is represented by the following integer array:

```
int matchingStatus[MAX_WORDLIST];
```

If an array element is 0, the corresponding word has not been matched. If an array element is 1, the corresponding word has already been matched. For example, suppose **listSize** is 3 and we have:

wordList[0] is "TRICK"	**matchingStatus[0]** is 0
wordList[1] is "OR"	**matchingStatus[1]** is 0
wordList[2] is "TREAT"	**matchingStatus[2]** is 1

That means the word "TREAT" has been matched while the words "TRICK" and "OR" have not.

Before the game starts, all the elements from **matchingStatus[0]** to **matchingStatus[listSize – 1]** are initialized to 0. When a word is matched later, the corresponding **matchingStatus** element will be set to 1.

4.6 Display Letter Grid and Word List

After initializing the matching status of the word list, the program will display the puzzle on the screen. You are required to complete and then invoke the following two functions from the **main** function:

- **`void printLetterGrid(char letterGrid[MAX_GRID][MAX_GRID], int gridSize)`**
  - Display the letter grid stored in **letterGrid** of size **gridSize**

- **`void printWordList(char wordList[MAX_WORDLIST][MAX_WORD], int listSize, int matchingStatus[MAX_WORDLIST])`**
  - Display the word list stored in **wordList** of size **listSize** with the matching status for each word (stored in **`matchingStatus`**)

An example is shown as follows, where all three words are not matched initially:

```
### 5 x 5 Letter Grid ###
  + 0 1 2 3 4
+ + + + + + +
0 + T T O E K
1 + Y A Z C M
2 + A E I X N
3 + I R Q R O
4 + T T U V J
### Word List of Size 3 ###
[ ] TRICK
[ ] OR
[ ] TREAT
```

Suppose the word “TREAT” is matched later, the following will be displayed:

```
### 5 x 5 Letter Grid ###
  + 0 1 2 3 4
+ + + + + + +
0 + T T O E K
1 + Y A Z C M
2 + A E I X N
3 + I R Q R O
4 + T T U V J
### Word List of Size 3 ###
[ ] TRICK
[ ] OR
[X] TREAT
```