CSE 2421 – Systems 1

Spring Semester 2024

Programming Assignment #3

- **This assignment is worth 12pts, and an optional additional point towards the midterm.**
- **You must upload the zipped solution folder to Carmen, as solution.zip.**
- **The deadline for this assignment is February 21st 11:59pm ET.**
- **Deductions for late submissions apply.**
- **The main topic of this assignment is static arrays, pointers and strings handling.**

Preliminary Instructions:
- Download the entire folder a3 from Carmen.
- You can debug and do the preliminary implementation of your code in your own laptop or desktop, but the final testing of your code must be done and work in stdlinux or coelinux.
- After downloading the folder a3 (or the file a3.tar.gz), copy the entire folder to your favorite OSU linux cluster (stdlinux or coelinux).

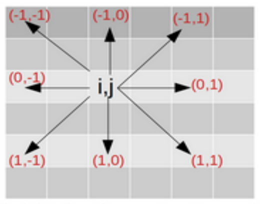
Instructions:

In this assignment you will write a "word find" (or "word search") program. You should only work in the file **words.c**.

The goal in the "word find" game is to search for a given word in a table (or matrix) of words. In the example below, the word "aaaa" is searched for in a matrix of 4x4 letters:

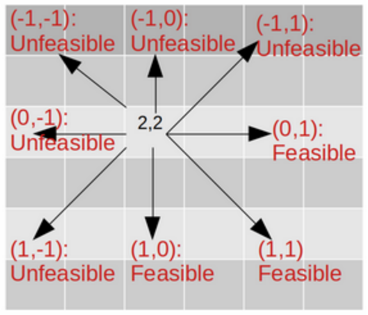
a	a	a	a
a	b	b	a
a	b	a	a
a	a	a	a

For example, searching for the word "aaaa" should return 8 matches, two along each border.
Words can start in any cell and be read in any of 8 directions.



All eight direction vectors to consider.

In our program, we wish to consider 8 directions to search for a word. That means we can read the word from left-to-right, but also right-to-left, up-to-down, down-to-up and in all diagonals. The figure above shows all eight direction vectors to consider given an initial coordinate in the matrix of words. Naturally, in order to consider the word found, the entire word should fit within the matrix of words and all of the letters should match the `needle` string. Consider the below example wherein any 4-letter word, starting at coordinate (2,2) could only exist in direction (0,1), (1,0) and (1,1).



Searching for a 4-letter word from coordinate [2,2], in a 6x6 matrix of words, renders only 3 feasible directions: (0,1), (1,0) and (1,1).

Next, we provide you with instructions to implement the "word find".

[3 pts] Implement function **`extract_word`**. The function should follow the signature below. This function loads a candidate string of the same length as **`needle`** from the matrix and returns it in the **`candidate`** argument. The string returned should start at the coordinate (`row,col`) of **`wordmat`** and be read along the direction vector (`di,dj`). The function should return 0 upon success, and return -1 when any part of the tentative string to be returned falls out of the bounds of the matrix of words. Notice that `extract_word` does not compare strings, it only extracts one word from the matrix and returns it:

```
int
extract_word (char wordmat[MAXN][MAXN], // Matrix of words.
  int nrows, int ncols, // Number of rows and columns in the matrix of words.
  int row, int col, // Coordinates of starting cell.
  int di, int dj, // Direction along rows and columns.
     // Each of di and dj can be one of {-1,0,1}.
  char needle[MAXN], // String to search for.
  char candidate[MAXN])
```

[4 pts] Implement a function **`search_one_word`**. The function should follow the signature below and return the number of matches of the argument **`needle`** found in the matrix of words.

```
int
search_one_word (
  int wordmat[MAXN][MAXN], // Matrix of words to search in.
  int nrows, int ncols, // Size of matrix of words.
  char *needle) // String to search for in the matrix of words.
```

The needle string can be found along all possible 8 directions from any matrix cell within bounds. The 8 possible directions result from all the combinations of {-1,0,1} x {-1,0,1} where at least one direction component is non-zero. Hence, the 8 possible directions for reading and finding words are:

Direction Vector (di,dj)	Description
(-1,-1)	up & left
(-1,0)	up
(-1,1)	up & right
(0,-1)	left
(0,1)	right
(1,-1)	down & left
(1,0)	down
(1,1)	down and right

Searching for the **`needle`** string involves testing all possible valid candidates along all 8 feasible directions. For all tests, you should first extract a **`candidate`** string using the **`extract_word`** function, and then compare the candidate string against the **`needle`** string by using the **`strcmp`** function defined in string.h. For convenience, the **`strcmp`** function signature is shown below. It is strongly recommended to test the function below in a standalone short program to understand how it works:

```
int strcmp(const char *s1, const char *s2);
RETURN VALUE: The  strcmp()  and strncmp() functions return an integer less than,
equal to, or greater than zero if s1 (or the first n bytes thereof) is found,
respectively, to be less than, to match, or be greater than s2.
```

You are also strongly encouraged to query the manual in your Linux system by issuing the following command in the terminal: `man 3 strcmp`

If you prefer to not use C's `strcmp` function you will have to implement your own string comparison function.

[3 pts] Implement the function **`search_words`**. The function should follow the signature below. The function should read from standard input the number of words **`nwords`** to search for. After this, the function should proceed to read **`nwords`** words (each an string), search for them in the matrix of words, and print the number of matches (occurrences) of that word in the matrix of words. **`search_words`** should use the function **`search_one_word`** previously implemented:

```
void
search_words (int wordmat[MAXN][MAXN], int nrows, int ncols)
```

This function is meant to process the input test files "input01.txt", "input02.txt" and "input03.txt". To test, for example, the file input02.txt do the following in the terminal (after building your binary):

```
cat input02.txt | ./search-words.test
```

[1 pt for Midterm Exam (Optional)] The initial codebase uses an implementation of function **`read_word_matrix`** which reads the matrix of words line-by-line using the `fgets` function. To get 1pt awarded in the midterm you must write an alternative implementation of **`read_word_matrix`** that reads the input matrix of words, character by character, from the standard input. Your implementation must use the macro `isspace` (defined in ctype.h) and the functions `getchar` and `ungetc` defined in `stdio`. The reason for using these two functions is the special handling needed to process whitespace characters and the newline character ('\n'). When writing your version of **`read_word_matrix`**, take into account the following:

- The shape of the matrix of words is given with two integer numbers in the first line of the input. The first number is the number of rows, and the second is the number of columns of the matrix of words.
- Next you must read the entire matrix, according to the number of rows and columns read initially.
- You are reading character-by-character. Remember that not all characters are meant to be stored in the matrix of words. In particular, characters that are not letters should be ignored.
- Every letter in the input of matrix of words is proceeded by a space character ' ' (Ascii code #32).
- After reading the matrix, your prompt (cursor) should be ready to read the number of words to search for. However, the read_word_matrix should not read this number.
- To understand how `ungetc` works use: `man 3 ungetc`
- To understand how `isspace` works use: `man 3 isspace`
- To test your version of read_word_matrix, do the following:
  ```
  make read-matrix.testplus
  cat input03.txt | ./read-matrix.testplus
  ```
- You can use the print-ascii-codes.test binary to view the structure of the input file in Ascii encoding:
  ```
  make print-ascii-codes.test
  cat input03.txt | ./print-ascii-code.test
  ```

[2 pts] Add descriptive comments to your code. Use descriptive variable names.

<u>What is provided and what to do:</u>

- A Makefile is provided. At this point you should know how to use it.
- Ideally, you should implement the functions in the order that they are requested.
- If you are unsure of how to use some C-function, search its definition in the manual (man 3 something).

- Several test harnesses are provided. These are the files test-*c.
- You should test the functions individually. The provided Makefile allows you to test each function independently. Read the Makefile to see what is being done. For example, to test `read-matrix` do:

```
make read-matrix.test
cat input02.txt |  ./read-matrix.test
```

- All other functions can be tested in a similar fashion.
- You can use “make clean” to remove object files and *.test files.
- When building binaries with the provided Makefile, only produce targets with extension .test and .testplus
- **Write your code exclusively in the file words.c.**
- Three text files with multiple test cases are provided. These are input01.txt, input02.txt and input03.txt.
- Do not leave additional printf calls or any extra printing message in words.c.
- Submitting your work: Create a folder called “solution” in your local machine. Copy the words.c file with your work and the screenshot file to the folder. Compress the folder and upload it to Carmen.

Where to upload:
Upload via Assignments->Assignment 3 in Carmen.
Do not confuse with Files->Assignments.