C Language Warm-up

We will be using the C programming language this course. Since you have never used C in any other course, it is a good thing for you to learn it in this course. C is so much similar to C++, so don't panic. However, there are some differences between C and C++ that you have to be aware of.
For this I recommend the following:

- Navigate to this link https://www.javatpoint.com/first-c-program
- You will find in the leftmost frame a list of mini-tutorials related to C. (control statements, structures, arrays, pointers, ..etc). Make sure to understand the following very well:
    - **C Tutorial:** Skim read the first four topics. Start from *First C Program* to the end of the tutorial (read properly).
    - **C control statements:** Similar to C++. No main differences. You can skim-read it.
    - **C Functions:** Similar to C++. No main differences. You can skim read the first three topics. Read **Storage Classes** properly.
    - **C Array: Read it properly.**
    - **C pointers: Read it properly.**
    - **C Dynamic Memory: Read it properly. (one topic)**
    - **C Strings: Read it properly. (They are very short).**

(1) Practice:

Open our Google Drive, and navigate to the folder **Introduction to C**. inside Lab 3. You will find a presentation summarizing all these points, read it properly after you finish these tutorials. You will also find some C snippet codes. Run them and make sure you understand them.

(2) Running C programs from the terminal:

1. Write your program in a normal text file and rename it program.c (you must give it an extension .c)

2. Open the terminal in the directory where program.c exists, (or cd to that directory in terminal).
3. Build (Compile) the c file by typing the following command:
   `gcc program.c -o program.out`
   gcc: is the build command
   program.c : is the name of your C file.
   program.o : The name of the output object file. If you didn't specify this option, it will default to a.out
4. Run the output object file by typing the following command:
   `./program.out`
   `or ./a.out`
5. Therefore to run any C program, you have to do TWO essential steps: BUILD AND RUN.
   - cd ../Desktop/CTutorial (the directory where program.C exists)
   - gcc program.c
   - ./a.out

   The last two lines can be replaced with:
   - gcc program.c -o program(or any other name).out
   - ./program.out
6. If gcc is not installed, type the following commands in the terminal:
   `sudo apt update`
   `sudo apt install build-essential`
   `sudo apt-get install manpages-dev`
7. To validate that the GCC compiler is successfully installed use the `gcc --version` command which will print the GCC version:
   `gcc --version`
   Source:
   https://linuxize.com/post/how-to-install-gcc-compiler-on-ubuntu-18-04/

(3) Running C programs from VS Code:

- Another alternative is to install VS Code in Ubuntu, and run your code in VS Code.
- There are tons of tutorials over the internet on how to setup your VS Code and run your C code in it.

Requirements:

Requirement #1:

Run the file “pointers.c” and answer the questions in the file. Submit a simple document containing answers to these questions.

Requirement #2:

A kangaroo word is a word that contains **ALL** the letters of one of its joey words; similar to the kangaroo who holds its joey in its pouch. A joey word is a word whose number of letters is **less than** that of the kangaroo word and whose letters appear all in the same order as they appear in the kangaroo word.

Examples on (kangaroo, joey) pairs are words like:

- chicken & hen
- lighted & lit
- instructor & tutor
- enjoyment -> joy

Given two strings **in no particular order**, you are required to find out if they are a Joey-Kangaroo Pair (JKP). Write a C program that takes two string arguments and prints 1 if they are a JKP or 0 if they aren’t. The input is provided from **the terminal** as two strings. Your program should be case-insensitive.

Notes:

1. A word and itself are not a JKP
2. You do not need to check on the validity of the inputs given in the problem. All the inputs are valid strings.
3. An example showing you how to read arguments from the terminal is attached with this file in **example.c**
4. Read the comments in the example file so you can understand how you should organize your code. Compile and run to understand how the input and output is expected.
5. Do not read any values from the user using “scanf” or “gets”.

Examples:

Suppose your file name isJKP.c, then you should compile and run as follows:

```
gcc isJKP.c -o isJKP
./isJKP supremacist racist
>>1
./isJKP bAd suBstandard
>>1
./isJKP Happy glad
>>0
./isJKP sad sad
>>0
./isJKP guy lad
>>0
```