1. Compare the conveniences and difficulties in implementing in COBOL and C.

   – Reading file in certain format
   In COBOL, it is more convenient to read some file in certain format. As we have defined the structure in the `FILE SECTION` before, when we read a line from that file, we can quickly use those data to do our task, e.g. comparison.

```
(...)
000280 FD INSTRUCTORS.
000281*CREATE STRUCTURE OF COURSES
000290 01 INS-RECORD.
000300     03 COURSE-CODE PIC X(5).
000310     03 REQUIRED-SKILLS.
000320         05 RSKILL1 PIC X(15).
000330         05 RSKILL2 PIC X(15).
000340         05 RSKILL3 PIC X(15).
000350     03 OPTIONAL-SKILLS.
000360         05 OSKILL1 PIC X(15).
000370         05 OSKILL2 PIC X(15).
000380         05 OSKILL3 PIC X(15).
000390         05 OSKILL4 PIC X(15).
000400         05 OSKILL5 PIC X(15).
(...)
```

Listing 1: A predefined structure for the data of instructor.txt

On the other hand, C does not have the concept of structured input, that means we need to separate the content manually to retrieve our 'parts' for later tasks.

```
while(fgets(insbuffer, sizeof(insbuffer), fins) != NULL){
(...)
    /*
      extracting the substring in a course record
    */
    cCode = strndup(insbuffer, 5);
    cReqSkill1 = strndup(insbuffer + 5, 15);
    cReqSkill2 = strndup(insbuffer + 20, 15);
    cReqSkill3 = strndup(insbuffer + 35, 15);
    cOptSkill1 = strndup(insbuffer + 50, 15);
    cOptSkill2 = strndup(insbuffer + 65, 15);
    cOptSkill3 = strndup(insbuffer + 80, 15);
    cOptSkill4 = strndup(insbuffer + 95, 15);
    cOptSkill5 = strndup(insbuffer + 110, 15);
(...)
}
```

Listing 2: The way to retrieve data in C.

Therefore, it is easier for COBOL reading file in certain format.

- Simulating loops

  For simulating loops, C have done a better job, in this assignment, we can do all the required tasks in two while loop.

```
while(fgets(insbuffer, sizeof(insbuffer), fins) != NULL){
(...)
    while(fgets(canbuffer, sizeof(canbuffer), fcan) != NULL){
    (...)
    }
}
(...)
```

  Listing 3: Showing how the task is done in C.

  Bun in COBOL, as we are restricted to use `GO TO` and `PERFORM` only, the logic become confusing in the later stage if you have multiple procedures, and simulating loop become harder as we need to trace from procedures.

```
(...)
001260          GO TO CHECK-SECOND-REQUIRED
(...)
001271      GO TO TA-SELECT-AND-RANKING.
(...)
001320          GO TO CHECK-THIRD-REQUIRED
(...)
001331      GO TO TA-SELECT-AND-RANKING.
(...)
001380          GO TO ADD-AND-RANK
(...)
001391      GO TO TA-SELECT-AND-RANKING.
(...)
```

  Listing 4: The logic flow of COBOL version of the ranking system.

  Therefore, in terms of the effect in simulation loops, C does a better job.

– Procedure/function call

In COBOL, we don't need to take care of pre-declaring the procedure, we can declare them on the fly.

```
001060 TA-SELECTION.
(...)
001170 TA-SELECT-AND-RANKING.
(...)
001219*CHECK IF MET REQ SKILLS
(...)
001279 CHECK-SECOND-REQUIRED.
(...)
001339 CHECK-THIRD-REQUIRED.
(...)
001399 ADD-AND-RANK.
(...)
```

Listing 5: Procedure calls in COBOL.

While in C, either 1) we need to put the function at the front of the program, or 2) declare it first, then we can type our function elsewhere, in this assignment, I have used method 2.

```
void output(FILE *fins, FILE *fcan);
void checkInsEmpty(FILE *fins, FILE *fout);
void checkCansEmpty(FILE *fins, FILE *fcan, FILE *fout);
int main(int argc, char const *argv[]) {
(...)
}
void output(FILE *fins, FILE *fcan){
(...)
}
void checkInsEmpty(FILE *fins, FILE *fout){
(...)
}
void checkCansEmpty(FILE *fins, FILE *fcan, FILE *fout){
(...)
}
```

Listing 6: Program declaration using method 2.

```
void output(FILE *fins, FILE *fcan){
(...)
}
void checkInsEmpty(FILE *fins, FILE *fout){
(...)
}
void checkCansEmpty(FILE *fins, FILE *fcan, FILE *fout){
(...)
}
int main(int argc, char const *argv[]) {
(...)
}
```

Listing 7: Program declaration using method 1.

Therefore, it will be easier to use COBOL for procedure/function call.

2. Compare COBOL with modern programming languages.
   Java, a multi-paradigm language, which is using object-oriented and imperative programming, have a different nature to COBOL, a imperative programming language.

   – Variable declarations
     Talking about variable declarations, Java, and many other modern programming language (e.g. Python, C++) do not need to pre-declare variables, if we want to declare a variable, we can do it anywhere we need it. In COBOL, we have to first define the variable we are going to use in `WORKING-STORAGE SECTION`, which will be a bit inconvenient as we have to plan the variable we have to use beforehand, and cannot declare them in `PROCEDURE DIVISION`.

   – Data type
     For most of the modern programming languages, including Java, data type have a certain size, e.g. `int` uses 4 bytes, `long` uses 8 bytes. This does not happen in COBOL. In COBOL, size of a variable have to be defined by the programmer. Take the error message ‘non-existing file!’ used in assignment 1 for example, in Java, the declaration would be `String errormsg = "non-existing file!";`, we do not have to specify the length, or the size of the string, but in COBOL, the declaration would be `000000 01 ERROR-MSG PIC X(18) VALUE 'non-existing file!'.`, we have to explicitly declare how many character we are going to use.

   – Parameter parsing
     In most of the modern programming languages, including Java, when we are using subroutine, we can pass the variable we need into the subroutine, and use them in that subroutine. In COBOL, there is no parameter parsing, we don’t need to parse any variable as the variable is globally useable.

   – Formatting
     Most of the modern programming languages, including Java, does not need to write in certain format. In COBOL, there is a certain format to follow, and there might be errors if we don’t follow this format.

3. Do you think COBOL is suitable for writing applications like in this assignment?

   – Programming difficulty
     In COBOL, when we need to read a file again from beginning, we have to close the file and open it again, this will cause a potential problem, is that if we close the file but we didn’t open the file, we might get an incorrect result from the program. With this action to read the file from the start again, we have to keep in mind that we need write proper statement. and this will add a little burden to a programmer.

   – Efficiency of you program
     As mentioned above, when we need to read a file again from beginning, we have to close the file and open it again. We have do extra check to see if we have meet the `EOF` of the file, this check will undoubtedly decrease the efficiency of our program.

4. In your program design, how do you separate the tasks into submodules?
   Tell us briefly the functionality of each submodule and the main flow of your program in terms of these submodules.

   – C Version

     In C version of the ranking system, I've separated into 3 submodules, which is `output`, `checkInsEmpty` and `checkCansEmpty`.

     - `output` is the submodule that do all the tasks if both instructors.txt and candidates.txt are not empty.
     - `checkInsEmpty` is to check if the instructors.txt file is empty, if it is, output the corresponding output.txt file for output purpose, if it isn't return to the caller.
     - `checkCansEmpty` is to check if the candidates.txt file is empty, if it is, output the corresponding output.txt file for output purpose, if it isn't return to the caller.

     Open File → Check the availability of the input files → If any one of the input file does not exist, terminate the program, else check the contents of the input → If the instructors.txt file is empty, do (a) → If the instructors.txt file is not empty but the candidates.txt file is empty, do (b) → if both file are not empty, do (c).

     (a) If the instructors.txt file is empty,
     Open the output.txt file → Output a empty file → Close the output.txt file → Terminate the program.

     (b) If the candidates.txt file is empty,
     Open the output.txt file → Read the instructors.txt file → Print the record of the course with Rank-k TA with `0000000000␣`. → Repeat until instructors.txt file EOF → Close the output.txt file → Terminate the program.

     (c) If both file are not empty,
     Open the output.txt file → Read the instructors.txt file → Extract the corresponding information → Read the candidates.txt file → Extract the corresponding information → Evaluate and rank the TA → Repeat until candidates.txt file EOF → Print the record with corresponding TA into the output.txt file→ Repeat until instructors.txt file EOF → Close the output.txt file → Terminate the program.

     We first open the file instructors.txt and candidates.txt in the `main` function, we check them is they can open correctly, if any of them cannot be open appropriately, we terminate the program and display the error message 'non-existing file!', if both of them can be open correctly, we pass them into the `output` function.
     In the `output` function, we first call the `checkInsEmpty`, then `checkCansEmpty`, if those function calls return back to `output`, we first read the content of instructors.txt, then we start to calculate the rankings of the TA listed in candidates.txt file for each couese, then we output the record, if all courses are all processed, we return to the `main` function, and terminate the program.

– <u>COBOL Version</u>
  In COBOL Version, there are numerous submodules.

  - `INSTRUCTORS-EXIST` Check if instructors.txt exist.
  - `CANDIDATES-EXIST` Check if candidates.txt exist.
  - `INSTRUCTORS-EMPTY` Check if instructors.txt is an empty file.
  - `CANDIDATES-EMPTY` Check if instructors.txt is an empty file.
  - `OUTPUT-WITH-PLACEHOLDER` Output the files with the course with Rank-k TA with `0000000000␣` if candidates.txt is empty.
  - `TA-SELECTION` Equivalent to the outer `while` loop of the C version of the program.
  - `TA-SELECT-AND-RANKING` Equivalent to the inner `while` loop of the C version of the program.
  - `CHECK-SECOND-REQUIRED` Check if second required skill is met.
  - `CHECK-THIRD-REQUIRED` Check if third required skill is met.
  - `ADD-AND-RANK` Evaluate the score and rank of a TA.
  - `INSERT-FIRST` Auxiliary function to insert TA into the first place of the table.
  - `INSERT-SECOND` Auxiliary function to insert TA into the second place of the table.
  - `INSERT-THIRD` Auxiliary function to insert TA into the third place of the table.
  - `EXIT-PROGRAM` Equivalent to `exit(1)` or `return 0` in C version.

  We first open the file in `MAIN-PARAGRAPH`, then go to `INSTRUCTORS-EXIST` to check if instructors.txt exist, if it does not exist, it display an error message and go to `EXIT-PROGRAM`; if it exist, it will go to `CANDIDATES-EXIST` to check if candidates.txt exist, if it does not exist, it display an error message and go to `EXIT-PROGRAM`; if it exist, it will go to `INSTRUCTORS-EMPTY` and check if instructors.txt is an empty file, if it is, it will output a empty file, then go to `EXIT-PROGRAM`; if it is not, it will go to `CANDIDATES-EMPTY` and check if instructors.txt is an empty file, if it is, it will first go to `OUTPUT-WITH-PLACEHOLDER` to output the corresponding output file, then go to `EXIT-PROGRAM`. If all inputs are not empty, it will go to `TA-SELECTION`, read and set the variables to a desired value, then go to `TA-SELECT-AND-RANKING`, read and set the variables to a desired value and start to evaluate the TAs, `CHECK-SECOND-REQUIRED` and `CHECK-THIRD-REQUIRED` will be subsequently called, it all of the check are passed, it will go to `ADD-AND-RANK`, in there, the points of each TA will be calculated, and `INSERT-FIRST`, `INSERT-SECOND` and `INSERT-THIRD` will be called to help putting the record of the TA to the appropriate table. We will then jump back to `TA-SELECT-AND-RANKING` to evaluate the next TA in the file, after the all TAs had been ranked, we will print the result to the output file, then we will jump back to `TA-SELECTION`, and rank the TAs for the next course, after all courses' TA have been ranked, we will go to `EXIT-PROGRAM` and terminate this program.