ECE220: Computer Systems & Programming **Spring 2024 ZJUI**
Machine Problem 4 **due: Saturday 16 March, 11:59:59 p.m.**

The Swim Meet

Your task this week is to use C to plan a sports event in an efficient manner. ZJUI is planning to hold a swimming competition (a "meet") in which a large number of students race against one another. All swimmers must swim a competitive trial in one of two lanes in the main pool. Unfortunately, many of the swimmers are picky and will whine and complain unless they are allowed to swim in their favorite lane, so the campus needs your help to plan out a schedule. The swimmers have been broken into groups of four, so we refer to a group as members A, B, C, and D. Given each swimmers' lane preferences, your job is to come up with a plan for the minimum number of trials (from two to four) that obey all swimmers' lane preferences.

The objective for this week is for you to gain some experience with control structures in C, particularly with operators and conditional statements, as well as basic I/O.

The Task

You must write the C function specified by the following:

```
void plan_swim_meet (int32_t pref_A, int32_t pref_B, int32_t pref_C,
                     int32_t pref_D);
```

The four parameters provide the lane preferences for each of the swimmers. Each is encoded as a two-bit value from 1 to 3. The 1 bit indicates whether or not a swimmer is willing to swim in lane 1, and the 2 bit indicates whether or not a swimmer is willing to swim in lane 2. Since each swimmer must swim in one of the two lanes, the possible values for each swimmer range from 1 to 3.

Given these four values, your function must determine the minimum number of trials required for the four swimmers, order the trials (details later), and print a summary of the trials to the screen using **`printf`**.

For example, if none of the four swimmers is willing to use lane 2, your function must print "A/B/C/D" (without quotation marks) to indicate that four trials with only one swimmer each are required in the order specified. On the other hand, if swimmers A and B have compatible preferences, and swimmers C and D have compatible preferences, your function must print "AB/CD" (as before, without quotation marks) to indicate that two trials are required (first A and B together, then C and D together). The rules for determining how to group the swimmers are as follows:

- Two swimmers must only swim together in a trial if their preferences are compatible (with one of the two using lane 1 and the other using lane 2).
- You must use the minimum total number of trials: two, three, or four.
- Each trial must be printed using the capital letters of the swimmers involved in the trial in alphabetical order (NOT in lane order). For example, if A and B are to swim at the same time, your function must always print "AB" and never "BA." Trials must be separated by "/" and the full summary must be followed by a line feed ("\n" in C strings).
- Trials must be ordered alphabetically (based on the first of the two letters when two swimmers are involved). For example, if swimmers A and D are swimming together, and swimmers B and C are swimming together, your function must print "AD/BC."

From these rules, you can easily deduce that swimmer A must always swim in the first trial.

Please note that **these rules are requirements for receiving functionality credit**. If you insert additional characters, use the wrong characters, use the wrong order, or so forth, you simply will not receive points.

Pieces

Your program will consist of a total of three files:

`mp4.h`	This header file provides function declarations and a brief description of the function that you must write for this assignment.
`mp4.c`	The source file for your function (a skeleton file has been provided). Be sure that you understand how the **`#include`** directives work to connect your function to the **`main`** program (see **`mp4main.c`**).

A third source file is also provided to you:

`mp4main.c`	A source file that interprets commands and calls your function.

You need not read this file, although you are welcome to do so.

Finally, we have also provided a **`Makefile`** that enables you to simply type "**`make`**" when you want to compile your code to an executable.

Specifics

You should look through the code provided to you before you begin coding.

- Your code must be written in C and must be contained in a file named **`mp4.c`**. We will NOT grade files with any other name. **You may not make changes to other files.** If your code does not work properly without such changes, you are likely to receive 0 credit.
- You must implement the **`plan_swim_meet`** function correctly.
- You may assume that all four swimmers' preferences are between 1 and 3.
- Your routine's output must be EXACTLY correct. See the rules in the task.
- Your code must be well-commented. Follow the commenting style of the code examples provided in class and in the textbook.

Compiling and Executing Your Program

Once you have created the **`mp4.c`** file and written the **`plan_swim_meet`** function, you can compile your code by typing:

```
make
```

Notice that **`make`** invokes the **`gcc`** compiler for you and passes the "**`-g`**" and "**`-Wall`**" flags. The "**`-g`**" argument tells the compiler to include debugging information so that you can use **`gdb`** to find your bugs (you will have some). The "**`-Wall`**" argument tells the compiler to give you warning messages for any code that it thinks likely to be a bug. Track down and fix all such issues, as they are usually bugs. Also note that if your code generates any warnings, you will lose points.

The compiler first translates the **`.c`** source files into **`.o`** object files using the "**`-c`**" flag. In these commands, the "**`-o <file name>`**" arguments indicate the object file name. The "**`-o swimmeet`**" argument in the final command tells the compiler to name the resulting program "**`swimmeet`**."

If compilation succeeds, you can then execute the program. The **`swimmeet`** program takes four command line arguments:

```
./swimmeet <A's prefs> <B's prefs> <C's prefs> <D's prefs>
```

Each argument corresponds to one of the four swimmers' lane preferences, and must be a number from 1 to 3. If the arguments given are not valid, the code in **`mp4main.c`** responds without calling your function.

Testing Your Code

For this assignment, we have provided you with a complete set of correct outputs. The script **`run_all_inputs`** executes the **`swimmeet`** program on all 81 possible valid input combinations and writes both a note about the parameters passed and the output produced by your program into a file called **`results`**. You can then use the **`diff`** tool to compare your answers with the correct ones. Specifically, to run the script and then compare:

```
./run_all_inputs
diff -C 1 results results_gold
```

The "**`-C 1`**" arguments instruct **`diff`** to provide a line of context around any differences found. If your program is producing output of the correct form, you will see any incorrect outputs along with the inputs that produced them (given above the corrupt and correct outputs). If you are getting any output from the **`diff`** command, you should expect to lose some (or all) functionality points.

Simplifying through Analysis

At first, this problem may seem difficult. But if you think for a bit about the rules provided, the number of possible combinations of trials is quite small. In fact, some combinations that might seem possible at first can never actually occur.

In addition to developing correct code for this task, we want you to spend some time thinking about the problem so as to make your code simpler to write. If you need a hint to get started, try writing down some possible combinations of trials that you might expect to see and then go look through the **`results_gold`** file to see if they ever occur. If they do not, ask yourself: why not?

Specific requirements have been included in the rubric below to challenge you.

Grading Rubric

We put a fair amount of emphasis on style and clarity in this class, as reflected in the rubric below.

Functionality (35%)

- 35% - Function generates the correct output based on the swimmers' lane preferences.

Style (50%)

- 20% - No matching **`printf`** statements—any identical results are combined using program logic.
- 15% - Up to four local variables are used to record swimmer compatibility—for example, can swimmer A swim in the same trial with swimmer B. You do not need more than four of these, so full points only if you cover all conditions with four or fewer.
- 15% - Code uses no more than five conditional statements (**`if`** statements) to identify the correct output. (Each case of a **`switch`** statement counts as one **`if`** statement.)

Comments, Clarity, and Write-up (15%)

- 5% - introductory paragraph explaining what you did (even if it's just the required work)
- 10% - code is clear and well-commented, and compilation generates no warnings (note: any warning means 0 points here)

Note that some categories in the rubric may depend on other categories and/or criteria. For example, if your code does not compile, you will receive no functionality points. If you print trial summaries incorrectly, you will also receive no functionality points. Finally, if your code produces any warnings (even ONE), you will lose 10 points—fix them before you submit your code.