# Homework #1

RELEASE DATE: 2025/03/17
DUE DATE: 2025/04/06, 23:59 on iLearning 3.0

1. *You need to submit your code to the designated place on iLearning 3.0.*
2. *As for coding, C and C++ (without lib such as STL) is allowed.*
3. *Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.*
4. *For each question, there will be a testcase (e.g. the testcase for question 1 is* testcase1.txt*) Please note that the testcase you received may not necessarily to be the same as the testcase used for grading. Please consider edge cases.*
5. *Let your program read the corresponding testcase and print the result to a corresponding text file(e.g. the output for question 2 should be* output2.txt*).*
6. *For each question, your program file should be named* DS{problem number}_{student ID}.c. *(e.g. assume your student ID is* 7112056067 *and you're solving question 3, your program file should be* DS3_7112056067.c)
7. *Each question is scored independently, but partial credit is not awarded; full credit is given only for complete correctness.*
8. *If you have any questions, please feel free to email the TA at any time.*

Teaching Assistant:

- Name: 王璽銘
- Email: g113056028@mail.nchu.edu.tw
- Lab: WCCCLab (S901)

# 1 Problem 1

You are given a string formula representing a chemical formula, and your task is to return the count of each atom in the formula

## Rules

1. An **atomic element** starts with an **uppercase letter**, followed by **zero or more lowercase letters**, representing the element's name.
2. If an element is followed by **one or more digits**, the digits represent the **count of that element**. If no digits follow, the count is **implicitly 1**.
   - **Valid examples:** "C6H12O6", "NaCl"
   - **Invalid example:** "H1O2" (since "1" should not be explicitly written)
3. Two formulas concatenated together form a **valid formula**.
   - **Example:** "C2H5OHNaCl" is a valid formula.
4. A formula enclosed in **parentheses** $(\ldots)$ followed by an optional **count** $(n)$ is also a valid formula.
   - **Examples:**
     - "(NH4)2SO4" (each element inside is multiplied by 2)
     - "(C6H6)3" (expands to "C18H18")

## Output Format

Return the count of all elements in **sorted order** (alphabetically), formatted as:

- The **element name**
- Followed by its **count** (if greater than 1)
- Repeated for all elements in ascending order.

## Sample Input

```
C6H12O6
Mg(OH)2
K4(Fe(CN)6)
```

## Sample Output

```
C6H12O6
H2MgO2
C6FeK4N6
```

# 2 Problem 2

Write a program to simulate the process scheduling of an operating system. Assume that the system has only one CPU, and each process has a known **arrival time**, **execution time**, and **priority level**. The priority level is represented by a natural number, where a **higher number indicates a higher priority**.

## Rules

1. If a process arrives when the CPU is idle, it will **occupy the CPU until it completes**, unless a new process with a **higher priority** arrives. In this case, the new (higher-priority) process will **preempt** the CPU, and the previous process must wait.
2. If a process arrives while the CPU is executing a process with a **higher or equal priority**, the newly arrived process must **wait**.
3. Once the CPU becomes idle, if there are waiting processes, the process with the **highest priority** will be executed first.
    - If multiple processes have the same highest priority, the process that arrived **earlier** will be selected.

## Input Format

The input consists of multiple lines, the first line represents the number of test cases. Begins with number of task(s), and each case containing four natural numbers (each not exceeding $10^8$), representing:

- **Process ID**
- **Arrival Time**
- **Execution Time**
- **Priority Level**

Each process has a unique process ID. No two processes with the same priority level will arrive at the same time. The input is **sorted in ascending order** by arrival time. The waiting queue will never exceed **15,000** processes at any time.

## Output Format

For each process, output its **process ID** and **completion time** in the order they finish execution.

## Sample Input

```
2
5
1 1 50 1
```

```
2 10 40 2
3 20 30 3
4 30 20 4
5 40 10 1
8
1 1 5 3
2 10 5 1
3 12 7 2
4 20 2 3
5 21 9 4
6 22 2 4
7 23 5 2
8 24 2 4
```

## Sample Output

```
4 50
3 70
2 100
1 141
5 151

1 6
3 19
5 30
6 32
8 34
4 35
7 40
2 42
```

# 3 Problem 3

You are tasked with designing a **two-dimensional dynamic maze navigation system**. The maze consists of an $N \times M$ grid where each cell represents a **room** at coordinate $(x, y)$. The agent starts at a given initial position and must navigate to the goal room.

The maze can change dynamically over time:

- **Blocking and unblocking rooms**.
- **Adding teleportation portals** (allowing instant movement between two rooms).
- **Recomputing shortest paths**.

## Rules

1. The maze is represented as an $N \times M$ **grid**, where each room $(x, y)$ can be linked to its four adjacent rooms.

2. **A `portal` can only be constructed on `'.'`, so does `block`.**
3. **`unblock` can only be used on `'#'`.**
4. Some rooms may contain **teleportation portals**, allowing direct movement between two rooms.
5. If a `portal` is constructed, put a **P** on the maze.
6. Teleportation portals are bidirectional, and entering one will automatically transport the agent to the connected room.
7. When a `path` command is issued, the program should compute the shortest path.

### Example Struct Implementation in C

```
typedef enum {
    UP = 0,
    DOWN = 1,
    LEFT = 2,
    RIGHT = 3
} Direction;

// Define the Room structure
typedef struct {
    int x, y;
    int blocked;
    int portal_x, portal_y;
} Room;
```

# Input Format

## Initial Maze

- The first line contains two integers $N, M$:
  - $N$ $(1 \leq N \leq 1000)$: Number of rows in the maze.
  - $M$ $(1 \leq M \leq 1000)$: Number of columns in the maze.
- The next $N$ lines contain $M$ characters representing the initial maze:
  - `.` - Walkable room.
  - `#` - Wall (unwalkable).
  - `S` - Starting position (unique).
  - `G` - Goal room (unique).

### Operations

- The next line contains an integer $Q$ $(1 \leq Q \leq 1000)$, representing the number of operations.
- Each of the next $Q$ lines contains one of the following commands:

```
portal X1 Y1 X2 Y2
block X Y
unblock X Y
path
print
```

## Output Format

- If executing `path`, output the length (Source is included in the path length) of the shortest path from the **current position** to **G** as:

  ```
  (Sx,Sy) -> (x2,y2) -> ... -> (Gx,Gy)
  ```

- If no valid path exists, output **"NO PATH"**.
- If `block X Y` attempts to block **S or G**, output **"INVALID OPERATION"**.
- If `print` is executed, output the current maze grid.
- There will be a `\n` in the last line.

## Sample Input

```
6 6
######
#S...#
#.#..#
#....#
#..G.#
######
14
unblock 2 2
path
block 1 2
block 2 1
portal 1 3 4 2
path
unblock 1 2
unblock 2 1
block 2 1
path
print
```

```
block 1 1
block 4 3
block 1 3
```

## Sample Output

```
6, (1,1) -> (2,1) -> (3,1) -> (4,1) -> (4,2) -> (4,3)
NO PATH
5, (1,1) -> (1,2) -> (1,3) -> (4,2) -> (4,3)
######
#S.P.#
##...#
#....#
#.PG.#
######
INVALID OPERATION
INVALID OPERATION
INVALID OPERATION
```

# 4 Problem 4

## Problem Description

The City has a large circular plaza, and to enhance its aesthetics and provide illumination at night, the municipal government has decided to install decorative lights around the outer edge of the plaza. The electrical planning department has designated $n$ installation points, numbered clockwise from 1 to $n$.

Each position $i$ has a brightness value $B_i$, representing the aesthetic improvement gained by installing a light at that position. However, due to power limitations, two adjacent positions cannot have lights installed at the same time (positions $i$ and $i + 1$ are adjacent, and notably, positions 1 and $n$ are also adjacent).

The government has $m$ decorative lights available and requires that all of them be installed. Your task is to design an installation scheme that maximizes the total brightness value while ensuring no two adjacent positions have lights.

If it is impossible to install all $m$ lights under these constraints, output ”Error!”.

## Input Format

- The first line represents number of testcases.
- The first line of each testcase contains two positive integers $n$ and $m$.
- The second line of each testcase contains $n$ integers, where the $i$-th integer represents $A_i$.

## Output Format

Output a single integer representing the maximum achievable total aesthetic value. If it's impossible to plant all $m$ saplings, output "Error!".

## Sample Input

```
2
7 3
1 2 3 4 5 6 7
7 4
1 2 3 4 5 6 7
```

## Sample Output

```
15
Error!
```

## Notes

- For all data: $m \leq n$, $-1000 \leq A_i \leq 1000$.

## Hint

- Use Circular Doubly Linked List and Priority Queue.

## Submission File

Please submit the homework to iLearning 3.0 before the deadline. You need to upload your coding part as a single **ZIP** compressed file to iLearning 3.0 before the deadline. The zip file should be named **DS_???????_{student ID}.zip** (e.g assume your student ID is 7112056067, your zip file should be DS_???????_7112056067.zip). The zip file should contain no directories and only the following items:

- all the source code
- an optional README, anything you want the TAs to read before grading your code