Homework #1

RELEASE DATE: 2025/03/18
DUE DATE: 2025/04/08, 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

Description

Implement a first-in-first-out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue:

- `push(x)`: Pushes element `x` to the back of the queue.
- `pop()`: Removes the element from the front of the queue and returns it.
- `peek()`: Returns the element at the front of the queue.
- `empty()`: Returns `true` if the queue is empty, `false` otherwise.

Notes

- You must use only standard operations of a stack – which means only `push` to top, `peek`/`pop` from top, `size`, and `is empty` operations are valid.
- Depending on your programming language, stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only standard operations of a stack.
- `peek()` should print the element at the front of the queue, and `empty()` should print true/false.

Sample Input

```
push(1)
push(2)
empty()
peek()
pop()
peek()
pop()
empty()
```

Sample Output

```
false
1
2
true
```

2 Problem 2

Description

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack:

- `push(x)`: Pushes element `x` onto the stack.
- `pop()`: Removes the element on the top of the stack and returns it.
- `top()`: Returns the element on the top of the stack.
- `empty()`: Returns `true` if the stack is empty, `false` otherwise.

Notes

- You must use only standard operations of a queue – which means only `push` to back, `peek/pop` from front, `size`, and `is empty` operations are valid.
- Depending on your programming language, queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only standard operations of a queue.
- `top()` should print the element on the top of the stack, and `empty()` should print true/-false.

Sample Input

```
push(1)
push(2)
empty()
top()
pop()
top()
pop()
empty()
```

Sample Output

```
false
2
1
true
```

3 Problem 3

Description

Given the head of a singly linked list and two integers `left` and `right` where `left` $\leq$ `right`, reverse the nodes of the list from position `left` to position `right`, and return the reversed list.

Notes

- The number of nodes in the list is `n`.
- $1 \leq \texttt{n} \leq 500$
- $-500 \leq \texttt{Node.val} \leq 500$
- $1 \leq \texttt{left} \leq \texttt{right} \leq \texttt{n}$
- You must implement using a linked list; otherwise, no credit will be given.

Input Format

The input consists of multiple lines, the first line represents the number of test cases. The first line of a testcase contains two number `left` and `right`, and the second line is the nodes in the linked list.

Sample Input

```
2
2 4
1 2 3 4 5
1 1
5
```

Sample Output

```
1 4 3 2 5
5
```

4 Problem 4

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
```

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