BÀI TẬP KIỂM TRA CUỐI KỲ HỌC PHẦN IT3230E - DATA STRUCTURES AND ALGORITHMS LAB. Sử dụng kết hợp với kiểm tra vấn đáp	**HỌC KỲ 2020-2**
Nhóm chuyên môn: Cao Tuấn Dũng, Phạm Quang Dũng, Nguyễn Hữu Đức, Nguyễn Kiêm Hiếu	**BỘ MÔN**

COVID19	**MÃ ĐỀ: 001**

Thông tin về COVID19 được thu thập và lưu trữ trong cơ sở dữ liệu, mỗi bản tin bao gồm các trường thông tin sau:

- Ngày: 1, 2, 3, …
- Mã vùng (xâu bao gồm các ký tự a,…,z, A,…,Z và 0,…,9)
- Số lượng bệnh nhân được phát hiện ở vùng và trong ngày hiện tại

Thực hiện một dãy các thao tác (cập nhật và truy vấn thông tin) được mô tả trong bảng dưới đây:

STT	Lệnh (thao tác)	Mô tả
1	$Update <region_code> <day> <number_of_patients>	Cập nhật số bệnh nhân <number_of_patients> được phát hiện trong ngày <day> tại vùng <region_code> Nói cách khác, thao tác này sẽ chèn một bản ghi bao gồm 3 thông tin (<region_code>, <day>, <number_of_patients>) vào cơ sở dữ liệu nếu cặp (<region_code>, <day>) chưa

		tồn tại trong cơ sở dữ liệu ***Chú ý:*** *nếu cặp (<region_code>, <day>) đã tồn tại trong cơ sở dữ liệu, thì sẽ không thực hiện chèn thêm vào nữa* ***Chú ý***: nếu cặp (<region_code>, <day>) chưa tồn tại trong cơ sở dữ liệu thì có nghĩa trong ngày <day>, tại vùng <region_code> không có ca bệnh nào được phát hiện
2	$CountTotalPatients	Đếm và **in** ra stdout (trên dòng mới) tổng số bệnh nhân được phát hiện tính đến thời điểm hiện tại
3	$FindNumberPatients <region_code> <day>	Tìm và **in** ra stdout (trên 1 dòng mới) số lượng bệnh nhân được phát hiện trong ngày <day> tại vùng <region_code>
4	$CountNumberPatientsOfPeriod <start_day> <end_day>	Đếm và **in** ra stdout (trên 1 dòng mới) số lượng bệnh nhân được phát hiện (trong tất cả các vùng) từ ngày <start_day> đến ngày <end_day>
5	$CountNumberPatientsOfRegion <region_code>	Đếm và **in** ra stdout (trên 1 dòng mới) số lượng bệnh nhân được phát hiện trong vùng <region_code>

Biết rằng số lượng mã vùng khác nhau có thể lên đến 1000, và số ngày có thể lên đến 70 (ngày được đánh số từ 1, 2, …, 70).

Dữ liệu (Stdin)

- Mỗi dòng ghi một lệnh với định dạng được mô tả trong bảng trên, và kết thúc dữ liệu là 1 dòng chứa ***

Kết quả (Stdout)

- Mỗi dòng chứa kết quả được in ra (bởi các lệnh 2, 3, 4, 5 được mô tả trong bảng trên)

Ví dụ

Stdin	Stdout
$Update HN0001 4 20 $Update HN0002 2 40 $Update HN0001 2 30 $Update HN0002 1 10 $CountTotalPatients $Update HN0003 1 45 $CountNumberPatientsOfPeriod 1 3 $CountNumberPatientsOfRegion HN0002 $FindNumberPatients HN0003 2 $FindNumberPatients HN0003 1 ***	100 125 50 0 45

Chú ý: *Thời gian thực hiện tối đa cho mỗi bộ dữ liệu test là* ***1 giây****. Trong bài tập này, sinh viên có thể sử dụng các cấu trúc dữ liệu Mảng, Danh sách liên kết, cây nhị phân tìm kiếm. Tuy nhiên việc dùng mảng hoặc danh sách liên kết với tìm kiếm tuần tự có thể không mang lại hiệu quả khi kích thước dữ liệu đầu vào lớn.*

COVID19	PROJECT CODE: 001

Information about COVID19 is collected and stored in a database, each object consists of following information:

- Day: 1, 2, 3, …
- Region code (string of a,…,z, A,…,Z and 0,…,9): codes of regions must be pairwise different
- Number of patients (integer) reported

Perform a sequence of commands (update actions and queries) described as follows

Index	Command	Description
1	$Update <region_code> <day> <number_of_patients>	Update information about new number of patients <number_of_patients> reported on day <day> in the region <region_code> In other words, this action inserts a record (<region_code>, <day>, <number_of_patients>) into database if the pair (<region_code>, <day>) does not exist ***Note:*** *if the pair (<region_code>, <day>) exists, then do not update*
2	$CountTotalPatients	Count and **print** to stdout (in a new line) the total number of patients reported until the current moment
3	$FindNumberPatients <region_code> <day>	Find and **print** to stdout (in a new line) the number of patients reported on day <day> in the region <region_code>
4	$CountNumberPatientsOfPeriod <start_day> <end_day>	Count and **print** to stdout (in a new line) the number of patients reported in all

		regions from day <start_day> until day <end_day>
5	$CountNumberPatientsOfRegion <region_code>	Count and **print** to stdout (in a new line) the number of patients reported in the region <region_code>

Number of distinct region codes can be upto 1000, and the number of days can be upto 70.

Input (Stdin)

- Each line contains a command with above format, terminated by a line ***

Output (Stdout)

- Each line contains a result printed out (by commands 2, 3, 4, 5 described in the table above)

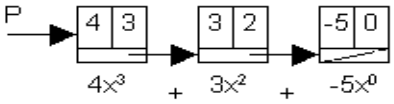
Example

Stdin	Stdout
$Update HN0001 4 20 $Update HN0002 2 40 $Update HN0001 2 30 $Update HN0002 1 10 $CountTotalPatients $Update HN0003 1 45 $CountNumberPatientsOfPeriod 1 3 $CountNumberPatientsOfRegion HN0002 $FindNumberPatients HN0003 2 $FindNumberPatients HN0003 1 ***	100 125 50 0 45

Note: *Time limit allowed for each test is* ***1 second****. In this project, students can use arrays, linked lists, binary search trees for representing the database. But arrays and linked lists with a sequential search cannot perfrom efficiently when the size of test is large.*

TÍNH TOÁN ĐA THỨC VỚI DANH SÁCH MÓC NỐI ĐƠN	MÃ ĐỀ: 002

Để tiết kiệm không gian bộ nhớ, người ta dùng cấu trúc dữ liệu danh sách móc nối để biểu diễn các đa thức, trong đó mỗi nút của danh sách chứa hệ số (**coefficient) và số mũ**(**exponent)** của một hạng tử. Ví dụ đa thức $4x^3 + 3x^2 - 5$, có thể được biểu diễn như hình vẽ bên phải. Hãy viết chương trình thao tác với đa thức có hệ số và số mũ nguyên sử dụng các danh sách móc nối, trong đó mỗi nút trong danh sách có thể được biểu diễn ví dụ như sau:




```
typedef struct {
 int coef;
 int exp;
} elementtype;
```

```
typedef struct node {
 elementtype term;
 struct node * next;
} node;
```

Chương trình liên tục tương tác với người dùng thông qua chuỗi các lệnh mô tả trong bảng dưới đây, cho đến khi gặp lệnh ***. Mọi lệnh thao tác trên bất cứ đa thức nào đều cần đảm bảo danh sách móc nối biểu diễn đa thức có số nút tối thiểu. Các tham số có ý nghĩa như sau:

- <LIST_ID> là các số nguyên 1, 2, 3, 4 chỉ đa thức thứ nhất, đa thức thứ 2, đa thức tổng của đa thức 1 và 2, đa thức tích của đa thức 1 và 2.
- coef và exp là các giá trị nguyên.

Index	Command	Description
1	insert < LIST_ID > <coef> <exp>	Chèn hạng tử với hệ số coef và số mũ exp như là một nút **vào đầu** danh sách ứng với đa thức LIST_ID là 1 hoặc 2. Chú ý là người dùng có thể chèn hạng tử với cùng một giá trị số mũ nhiều lần. **Chương trình cần thao tác trên danh sách sao cho số nút trong danh sách là tối thiểu.**
2	display < LIST_ID >	Hiển thị đa thức <LIST_ID> trên một dòng dưới dạng một dãy các số nguyên là hệ số và số mũ ứng với từng hạng tử, mỗi giá trị cách nhau một dấu cách. Dãy kết thúc là một dấu cách sau đó xuống dòng. Ví dụ: 3<space>2<space>4<space>3<space>-5<space>0<space>

3	sort < LIST_ID >	Sắp xếp lại các nút trong danh sách móc nối sao cho nó biểu diễn đa thức với các hạng tử với **số mũ giảm dần.** Hiển thị đa thức sau khi sắp xếp như mô tả ở lệnh display (2) ra màn hình. Ví dụ: 4<space>3<space>3<space>2<space>-5<space>0<space>
4	add	Tính tổng hai đa thức 1, 2 và lưu trữ đa thức kết quả ứng với LIST_ID là 3 vào một danh sách móc nối. Hiển thị đa thức kết quả **sau khi đã sắp xếp** ra màn hình như mô tả ở chức năng display
5	multiply	Tính tích hai đa thức 1,2 và lưu trữ đa thức kết quả ứng với LIST_ID là 4 vào một danh sách móc nối. Hiển thị đa thức kết quả **sau khi đã sắp xếp** ra màn hình.
6	space < LIST_ID >	Hiển thị số byte mà danh sách tương ứng với đa thức < LIST_ID > chiếm trong bộ nhớ, biết rằng mỗi nút trong danh sách chiếm 10 byte (không quan tâm đến cách chương trình biểu diễn danh sách trên thực tế).
7	reset <LIST_ID>	Giải phóng toàn bộ danh sách ứng với đa thức <LIST_ID>. Đa thức được khởi tạo về trạng thái lúc bắt đầu chương trình.
8	***	Thoát vòng lặp đợi lệnh, giải phóng bộ nhớ đang được cấp phát cho các đa thức, và thoát chương trình.

Đầu vào (dòng vào chuẩn)

- Mỗi dòng nhập một lệnh với định dạng mô tả ở bảng trên, kết thúc với lệnh ***

Đầu ra (dòng ra chuẩn)

- Kết quả được hiển thị trên mỗi dòng màn hình với các lệnh có yêu cầu cần đưa ra kết quả.

Ví dụ

Stdin	Minh họa
insert 1 -6 3 insert 1 4 2 insert 1 1 0 insert 1 2 3 display 1 sort 1	L1: -6x^3 L1: 4x^2 - 6x^3 L1: 1 + 4x^2 - 6x^3 L1: 1 + 4x^2 -4x^3

insert 2 -1 1 insert 2 -2 2 display 2 add space 3 multiply ***	L2: -x L2: -2x^2 - x

Stdout	Minh họa
1 0 4 2 -4 3<space> -4 3 4 2 1 0<space> -2 2 -1 1<space> -4 3 2 2 -1 1 1 0<space> 40 8 5 -4 4 -4 3 -2 2 -1 1<space>	1 + 4x^2 -4x^3 -4x^3 + 4x^2 +1 -2x^2 - x L3: -4x^3+2x^2- x+1 40 byte L4:8x^5-4x^4-4x^3-2x^2-x

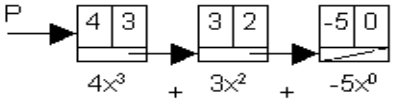
Sinh viên lưu ý:

Mọi lệnh thao tác trên bất cứ đa thức nào đều cần đảm bảo danh sách móc nối biểu diễn đa thức có số nút tối thiểu.

Để đơn giản hóa việc lập trình và tránh bị mất điểm khi codeforce so sánh chuỗi, chương trình phải hiển thị dấu space (cách) ở vị trí cuối cùng khi hiển thị đa thức. Testcase đề thi sẽ luôn có ký tự space ở cuối cùng.

Chương trình phải sử dụng danh sách móc nối (đơn hoặc đôi,..) để biểu diễn đa thức. Nếu sinh viên sử dụng cấu trúc dữ liệu khác sẽ bị trừ điểm.

POLYNOMIAL MANIPULATION WITH LINKED LISTS	PROJECT CODE: 002



To save memory space, a polynomial may be represented as a linked list where each node contains the **coefficient** and **exponent** of a term of the polynomial. For example, the polynomial $4x^3 + 3x^2 - 5$, would be represented as shown at right. Write a program that works with polynomials whose coefficients and exponents are integers using linked lists, where each node in the list can be represented as in the following example:

```
typedef struct {
 int coef;
 int exp;
} elementtype;
```

```
typedef struct node {
 elementtype term;
 struct node * next;
} node;
```

The program continuously interacts with the user through the sequence of commands described in the table below, until the command *** is encountered. The parameters have the following meanings:

- <LIST_ID> are integers 1, 2, 3, 4 indicating the first polynomial, the second polynomial, the sum polynomial of polynomials 1 and 2, and the product polynomial of polynomials 1 and 2.
- coef and exp are integer values.

Index	Command	Description
1	insert < LIST_ID > <coef> <exp>	Insert an element with coefficient coef and exponent exp as a node **at the beginning** of the list corresponding to the LIST_ID polynomial of **1 or 2**. It should be noted that the user can insert elements with the same exponent value more than once. **The program must operate on the list so that the number of nodes in the list is minimal.**
2	display < LIST_ID >	Display the polynomial <LIST_ID> on a single line as a series of integers that are the coefficients and exponents for each element, each value separated by a space. **The output ends with a space** followed by a newline. Eg:

		3<space>2<space>4<space>3<space>-5<space>0<space>
3	sort < LIST_ID >	Arrange (sort) the nodes in the linked list so that it represents a polynomial with elements with **decreasing exponents**. Display the polynomial after sorting as described in the display command to the screen. Eg: 4<space>3<space>3<space>2<space>-5<space>0<space>
4	add	Sum the two polynomials 1 and 2 and store the resulting polynomial with a LIST_ID of 3 in a linked list. Display the resulting polynomial **after sorting** to the screen as described in the display command
5	multiply	Calculate the product of the two polynomials 1,2 and store the resulting polynomial with a LIST_ID of 4 into a linked list. Display the resulting polynomial **after sorting** on the screen.
6	space < LIST_ID >	Displays the number of bytes that the list corresponding to the <LIST_ID> polynomial occupies in memory, given that each node in the list occupies 10 bytes (regardless of how the program actually represents the list).
7	reset <LIST_ID>	Free the list corresponding to the polynomial <LIST_ID>. The polynomial <LIST_ID> is initialized to the state at the start of the program.
8	***	Exit the command waiting loop, free the memory being allocated for the polynomials, and exit the program.

Input

- Each line contains a command with above format, terminated by a line ***

Output

- Each line contains a result printed out (by commands 1, 2, 3, 4, 5, 6, 7, 8 described in the table above)

Example for student

Stdin	illustration
insert 1 -6 3 insert 1 4 2	L1: -6x^3 L1: 4x^2 - 6x^3

insert 1 1 0 insert 1 2 3 display 1 sort 1 insert 2 -1 1 insert 2 -2 2 display 2 add space 3 multiply ***	L1: 1 + 4x^2 - 6x^3 L1: 1 + 4x^2 -4x^3 L2: -x L2: -2x^2 - x

Stdout	illustration
1 0 4 2 -4 3<space>	1 + 4x^2 -4x^3
-4 3 4 2 1 0<space>	-4x^3 + 4x^2 +1
-2 2 -1 1<space>	-2x^2 - x
-4 3 2 2 -1 1 1 0<space>	L3: -4x^3+2x^2- x+1
40	40 byte
8 5 -4 4 -4 3 -2 2 -1 1<space>	L4:8x^5-4x^4-4x^3-2x^2-x

NOTES:

All commands that operate on any polynomial needs to ensure that the linked list representing the polynomial has a minimum number of nodes.

To simplify programming and avoid losing points when codeforce compares strings, the program must display spaces at the end when displaying polynomials. The test case will always have a space at the end.

The program must use a linked list (singly or doubly, ..) to represent the polynomial. If students use other data structures, their marks will be deducted.

SIÊU THỊ	MÃ ĐỀ: 003

Một siêu thị tổ chức việc thanh toán cho khách hàng với bốn quầy thu ngân và bốn hàng khách (được đánh số 1,2,3,4). Hoạt động thanh toán được thực hiện như mô tả sau:

- Khi có yêu cầu thanh toán, một **nhân viên điều phối thông thường** sẽ hướng dẫn khách hàng vào cuối một trong các hàng theo nguyên tắc phân phối **round-robin** (gửi vào lần lượt từng hàng và quay vòng lại: VD: 1,2,3,4,1,2,3,4,...). Khách hàng được phát một **mã số** để tiến hành gọi khi checkout. Mã số khách hàng là một số nguyên, tăng dần cho từng khách hàng mới (bắt đầu từ 1).
    - **Nhân viên điều phối tốt** có thể hướng dẫn khách hàng vào hàng có ít người nhất. Việc điều phối này không ảnh hưởng đến thứ tự phân phối của nhân viên thông thường (hàng tiếp theo trong phân phối round-robin sẽ giữ nguyên).
- **Nhân viên thu ngân** tại quầy thanh toán thứ i sẽ gọi lần lượt các khách hàng trong hàng thứ i để thực hiện checkout.
- Tùy thuộc vào số lượng khách hàng mà siêu thị quyết định mở thêm một quầy thu ngân mới (cùng với hàng khách mới) hoặc đóng một quầy thu ngân.
    - Trong trường hợp mở thêm quầy thu ngân mới, quầy mới sẽ được đặt cuối cùng trong danh sách quầy. Số hiệu quầy mới và hàng tương ứng sẽ được đặt tăng dần, bắt đầu từ 5. Lưu ý hàng tiếp theo trong phân phối round-robin có thể thay đổi khi mở thêm quầy mới (VD. có 4 hàng và đã phân phối đến hàng thứ 4, khi thêm vào quầy mới vào cuối danh sách, hàng mới sẽ là hàng tiếp theo trong phân phối).
    - Trong trường hợp đóng một quầy thu ngân, số hiệu quầy thu ngân và hàng còn lại sẽ được **giữ nguyên**. Lần lượt từng khách hàng trong hàng tương ứng với quầy bị đóng(thứ tự từ đầu đến cuối hàng) sẽ được chuyển sang các hàng còn lại theo nguyên tắc round-robin, bắt đầu từ quầy tiếp theo trong phân phối này. Lưu ý hàng bị đóng có thể chính là hàng tiếp theo trong phân phối round-robin, khi đó cần dịch chuyển tiếp tục giá trị này trước khi đóng quầy.
      Chỉ được đóng quầy khi số lượng quầy >= 2.

Hãy viết một chương trình mô phỏng các hoạt động này. Chương trình có các chức năng chính như sau:

Index	Lệnh	Mô tả
1	**$Enter**	**Nhân viên điều phối thông thường** hướng dẫn khách hàng vào một trong các hàng theo

		nguyên tắc **round robin**. In ra ***mã số khách hàng*** và ***số hiệu hàng*** khách hàng đó được sắp xếp.
2	**$AdvancedEnter**	**Nhân viên điều phối tốt** hướng dẫn khách hàng vào hàng đợi có **ít người nhất**. In ra ***mã số khách hàng*** và ***số hiệu hàng*** khách hàng đó được sắp xếp. (Nếu có nhiều hàng có cùng một số lượng nhỏ nhất, chọn hàng có số hiệu thấp nhất).
3	**$Checkout** <cashier_number>	Nhân viên thu ngân tại quầy <cashier_number> tiến hành checkout cho khách hàng đầu tiên trong hàng tương ứng. In ra ***số hiệu khách hàng*** thực hiện checkout. (Nếu hàng đang rỗng, in ra "Empty". Nếu quầy <cashier_number> không tồn tại, in ra "Error")
4	**$CountNumberCustomerInLine** <line_number>	Đếm và in ***số lượng khách hàng*** tại hàng <line_number>. Nếu quầy <line_number> không tồn tại, in ra "Error".
5	**$ListCustomerInLine** <line_number>	In ra ***danh sách mã số khách hàng*** tại hàng <line_number> (theo thứ tự từ đầu đến cuối hàng). Nếu quầy <line_number> không tồn tại, in ra "Error".
6	**$CountNumberCustomerInAllLines**	Đếm và in ***số lượng khách hàng*** trong tất cả các hàng.
7	**$OpenCashier**	Mở thêm vào một quầy thu ngân mới và một hàng tương ứng. In ra ***số hiệu quầy*** mới được thêm.
8	**$CloseCashier** <cashier_number>	Đóng quầy thu ngân <cashier> và hàng tương ứng. In ra "Closed" nếu đóng thành công, hoặc "Error" nếu số lượng quầy hiện tại <= 1 hoặc quầy <cashier_number> không tồn tại.

9	***	Kết thúc chương trình và giải phóng bộ nhớ

Đầu vào (dòng vào chuẩn stdin):

- Mỗi lệnh được viết trên một dòng theo ký pháp trên với dòng kết thúc là ***

Đầu ra (dòng ra chuẩn stdout)

- Mỗi dòng chứa kết quả được in ra bởi các lệnh như mô tả trên.

Ví dụ

Stdin	**Stdout**	**Trạng thái các hàng (hàng tiếp theo trong phân phối round-robin được in đậm)**
$Enter $Enter $Enter $Checkout 2 $AdvancedEnter $Enter $Enter $CountNumberCustomerInLine 1 $ListCustomerInLine 1 $Checkout 1 $ListCustomerInLine 1 $CountNumberCustomerInAllLines $Enter $OpenCashier $CloseCashier 2 ***	1<space>1 2<space>2 3<space>3 2 4<space>2 5<space>4 6<space>1 2 1<space>6<space> 1 6<space> 4 7<space>2 5 Closed	1->[1]; **2**->[]; 3->[]; 4->[] 1->[1]; 2->[2]; **3**->[]; 4->[] 1->[1]; 2->[2]; 3->[3]; **4**->[] 1->[1]; 2->[]; 3->[3]; **4**->[] 1->[1]; 2->[4]; 3->[3]; **4**->[] **1**->[1]; 2->[4]; 3->[3]; 4->[5] 1->[1,6]; **2**->[4]; 3->[3]; 4->[5] 1->[1,6]; **2**->[4]; 3->[3]; 4->[5] 1->[1,6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; 2->[4,7]; **3**->[3]; 4->[5] 1->[6]; 2->[4,7]; **3**->[3]; 4->[5]; 5->[] 1->[6]; 3->[3,4]; 4->[5,7]; **5**->[]

MARKET PLACE	PROJECT CODE: 003

A supermarket manages its business by organizing four cashiers and four lines of customers (which are numbered as 1,2,3,4). The checking out process would be described as follows:

- When receiving a checkout request from a customer, a **normal coordinator** will guide the customer into a designated line with principles of round-robin distribution (i.e., customers will be sent one after one to lines: 1,2,3,4,1,2,3, 4, …). A customer code, which is an incremental number, starting from 1, will be given to the customer.
    - **A good coordinator** can guide customers to the line which has least number of customers. This activity doesn't affect to the guidance of the normal coordinator (i.e., the next line in round-robin distribution doesn't change).
- **The staff at the cashier number i** will call a customer in front of the line number i for checking out.
- According to the number of customers, supermarket will decide to open a new cashier or close an existing one (together with the corresponding line).
    - Open a new cashier: new cashier will be put at the end of the cashier list. Cashier number as well as its line number will be given as an incremental number, starting from 5. Please note that the next line in round-robin may have to chance due to of this insertion (Eg. There are 4 lines and the last round-robin guidance is to the 4^{th} line. When a new line is inserted at the end, next round-robin distribution should be that new line).
    - Close an existing cashier: all other cashiers keep their number. Customers in the closing line should be moved to the remaining lines with the same round-robin principle, starting from the same next round-robin line. Note that, the line to be closed may be the same next round-robin line. In this case, next round-robin line should be shifted before processing cashier close.

      A cashier can only be able to close if the total number of cashier >= 2.

Let write a C program to simulate the business of the marketplace. The program should have the following functionalities:

Index	Command	Description
1	**$Enter**	**Normal coordinator** guides a customer to the end of a customer

		line with round-robin principle. Print out ***customer code*** and ***line number*** which the customer heads to.
2	**$AdvancedEnter**	**Good coordinator** guides a customer to the end of the customer line which have least number of people. Print out ***customer code*** and ***line number*** which the customer heads to. (If there are more than one least-people lines, choose the line with the smallest number).
3	**$Checkout** <cashier_number>	The staff at the cashier <cashier_number> processes checking out for the customer in front of the corresponding line. Print out the customer code of the customer being checking out. (If this line is empty, print out "Empty". If there is no such cashier of the number <cashier_number>, print out "Error")
4	**$CountNumberCustomerInLine** <line_number>	Count and print number of customers in the line <line_number>. If the line <line_number> does not exist then print "Error".
5	**$ListCustomerInLine** <line_number>	Print list of customer codes for the customers in line (from the beginning of the line to the end of the line). If the line <line_number> does not exist then print "Error".
6	**$CountNumberCustomerInAllLines**	Count and print number of customers in all lines.
7	**$OpenCashier**	Open a new cashier and the coresponding line. Print the cashier number to be openned.
8	**$CloseCashier** <cashier_number>	Close the cashier

		<cashier_number> and its line. Print “Closed” at success, or print “Error” if number of the cashiers <= 1, or the cashier <cashier_number> does not exist.
9	***	Free memory and terminate the program.

Input (stdin):

- Each line contains a command with above format, terminated by a line ***

Output (dòng ra chuẩn stdout)

- Each line contains a result printed out as described above

Example

Stdin	**Stdout**	**State of lines (Next round-robin line is in bold text)**
$Enter $Enter $Enter $Checkout 2 $AdvancedEnter $Enter $Enter $CountNumberCustomerInLine 1 $ListCustomerInLine 1 $Checkout 1 $ListCustomerInLine 1 $CountNumberCustomerInAllLines $Enter $OpenCashier $CloseCashier 2 ***	1<space>1 2<space>2 3<space>3 2 4<space>2 5<space>4 6<space>1 2 1<space>6<space> 1 6<space> 4 7<space>2 5 Closed	1->[1]; **2**->[]; 3->[]; 4->[] 1->[1]; 2->[2]; **3**->[]; 4->[] 1->[1]; 2->[2]; 3->[3]; **4**->[] 1->[1]; 2->[]; 3->[3]; **4**->[] 1->[1]; 2->[4]; 3->[3]; **4**->[] **1**->[1]; 2->[4]; 3->[3]; 4->[5] 1->[1,6]; **2**->[4]; 3->[3]; 4->[5] 1->[1,6]; **2**->[4]; 3->[3]; 4->[5] 1->[1,6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; **2**->[4]; 3->[3]; 4->[5] 1->[6]; 2->[4,7]; **3**->[3]; 4->[5] 1->[6]; 2->[4,7]; **3**->[3]; 4->[5]; 5->[] 1->[6]; 3->[3,4]; 4->[5,7]; **5**->[]

NHÀ HÀNG	MÃ ĐỀ: 004

Xây dựng chương trình phục vụ gọi món tại quán cafe **Good Morning** giai đoạn "Bình thường mới"

Khách đến gọi đồ trong menu. Nhân viên sẽ nhập thông tin order vào hệ thống. Hệ thống dựa trên các món mà quan đang phải chuẩn bị để ước lượng thời gian chờ của khách.

Menu có các loại đồ uống (DRINK) và đồ ăn nhẹ (FOOD) và có giá cho trước. Quán có một bộ phận chế biến đồ uống riêng và một bộ phận chế biến đồ ăn nhẹ riêng. Mỗi bộ phận tại một thời điểm chỉ có thể chế biến một món duy nhất, nếu có nhiều món thì phải chờ lần lượt. Thời gian chế biến một món là cho trước. Quán phục vụ theo quy tắc khách order trước làm trước.

Chương trình có các chức năng sau:

1. $ADD <loại_món> <tên_món> <giá_tiền> <thời_gian_chế_biến>: Nếu món đã có trong menu sẽ giữ nguyên thông tin theo menu, nếu chưa có sẽ thêm vào menu. Sau đó, chương trình sẽ in ra số lượng món hiện có trong menu. Trong menu không có hai món nào có tên giống nhau. Loại món có một trong hai giá trị FOOD hoặc DRINK, tên món là chuỗi kí tự không chứa dấu cách.
2. $PRICE <tên_món>: Chương trình thông báo giá tiền của một món. Giả sử tên món này đã có trong menu.
3. $ORDER <giờ> <phút> <tên_món_1> <tên_món_2>: Tại thời điểm (giờ phút) nhập vào, khách vào quán order đúng hai món đồ khác nhau có trong menu (Giờ là số nguyên có giá trị từ 0 đến 23, phút là số nguyên có giá trị từ 0 đến 59). Chương trình in ra thời gian chờ theo phút để hoàn thành order của khách. Chú ý thời gian này bao gồm cả thời gian chế biến và thời gian chờ để quán hoàn thành các món trước đó đã được các khách hàng khác order (nếu có). Nếu món_1 và món_2 cùng loại, món_1 sẽ được ưu tiên chế biến trước.
4. $WAIT <giờ> <phút>: Theo thời điểm nhập vào, chương trình in ra số lượng món đang chờ hoàn thành.
5. $COUNT <tên_món>: Thống kê tổng số lần món ăn này đã được khách order. Giả sử tên món nhập vào đã có trong menu.
6. *** Chương trình kết thúc

Chú ý: Trong cùng một testcase, thời điểm đi kèm với các câu lệnh $ORDER và $WAIT sẽ luôn được đảm bảo tăng dần theo thứ tự xuất hiện của các câu lệnh này (nghĩa là không xảy ra trường hợp đã có câu lệnh $ORDER lúc 9 20 rồi sau đó lại có câu lệnh $WAIT lúc 9 2).

Testcase 1:

$ADD DRINK Ca_phe 30000 3 $ADD DRINK Nuoc_chanh 20000 3 $ADD FOOD Banh_ngot 25000 2 $ADD FOOD Huong_duong 25000 2 ***	1 2 3 4

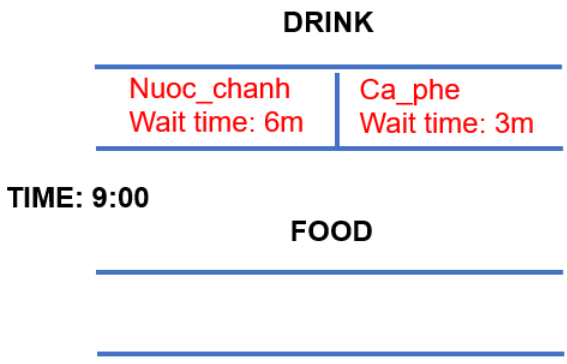
Testcase 2:

$ADD DRINK Ca_phe 30000 3 $ADD DRINK Nuoc_chanh 20000 3 $ADD FOOD Banh_ngot 25000 2 $ADD FOOD Huong_duong 25000 2 $PRICE Huong_duong ***	1 2 3 4 25000

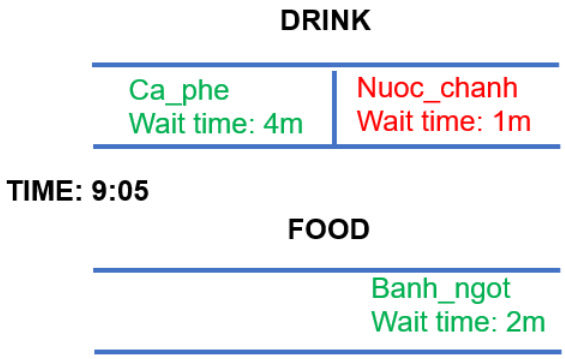
Testcase 3:

$ADD DRINK Ca_phe 30000 3 $ADD DRINK Nuoc_chanh 20000 3 $ADD FOOD Banh_ngot 25000 2 $ADD FOOD Huong_duong 25000 2 $ORDER 9 0 Ca_phe Nuoc_chanh $ORDER 9 5 Banh_ngot Ca_phe ***	1 2 3 4 6 4

$ORDER 9 0 Ca_phe Nuoc_chanh → 6



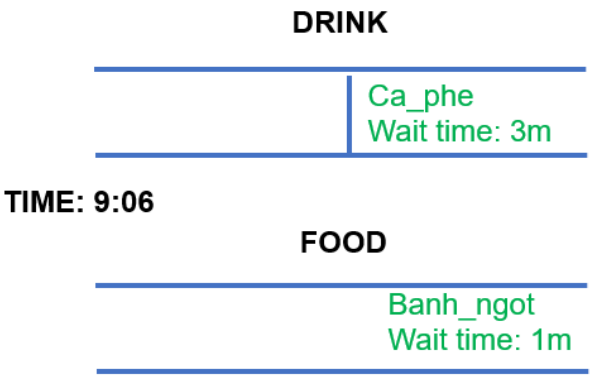
$ORDER 9 5 Banh_ngot Ca_phe → 4



Testcase 4:

$ADD DRINK Ca_phe 30000 3 $ADD DRINK Nuoc_chanh 20000 3 $ADD FOOD Banh_ngot 25000 2 $ADD FOOD Huong_duong 25000 2 $ORDER 9 0 Ca_phe Nuoc_chanh $ORDER 9 5 Banh_ngot Ca_phe $WAIT 9 6 ***	1 2 3 4 6 4 2

$WAIT 9 6 → 2



Testcase 5:

$ADD DRINK Ca_phe 30000 3 $ADD DRINK Nuoc_chanh 20000 3 $ADD FOOD Banh_ngot 25000 2 $ADD FOOD Huong_duong 25000 2 $ORDER 9 0 Ca_phe Nuoc_chanh $ORDER 9 5 Banh_ngot Ca_phe $COUNT Ca_phe ***	1 2 3 4 6 4 2