Задача Waiter

Header C++ `waiter.h`

Сервитьор, сервитьор, още един полипротонен буфер, моля!

О, не! Утре малкият `IR12661` *има класно по предмета принципи на химическата наука. Това го измъчва през целия последен месец, но тази вечер ще достигне връхната си точка и ще свърши (в различна часова зона). Сполетяват го ужасни кошмари за слаба киселина, силна основа и писмената работа, която трябва да предаде в рамките на* 12 *часа, влизащи заедно в един бар. Моля, помогнете му да се събуди, така че да започне първата чернова на работата си преди да е твърде късно.*

В момента сме в любимото място на българина – ресторант с храна на корем, като има точно четири главни герои: `Киселин` и `Основен` — двамата клиенти; `Зотеро` — главния готвач; и разбира се сервитьор робот.

Известно е, че менюто включва N ястия, а клиентите имат много различни предпочитания относно храната (естествено). Затова може да разглеждаме предпочитанията като множество $S \subset \{1, 2, \ldots, N\}$ и неговото допълнение, като `Киселин` предпочита всички храни от S, а `Основен` предпочита всички храни от $\{1, 2, \ldots, N\} \setminus S$. Гарантирано е, че $0 < |S| < N$.

Начинът, по който двамата клиенти могат да поръчват храна, е както следва: на всеки десет минути, сервитьорът **произволно** (вижте детайлите по имплементацията) избира клиент и изпълнява тяхната поръчка. Клиентът може да поиска едно нещо от менюто (част от списъка с предпочитания), **или** да каже, че не иска нищо засега. Сервитьорът изпраща поръчката в кухнята — съобщава я на `Зотеро`, след което по същия начин продължава цикълът.

Обърнете внимание, че `Киселин` и `Основен` са в едно и също помещение, така че чуват поръчките си.

Оказва се, че `Киселин`, `Основен` и `Зотеро` са добри приятели от едно време, но в момента са затворени от сервитьора в ресторанта, който няма да ги пусне, докато главният готвач, `Зотеро`, не открие предпочитанията на клиентите. В този дух задачата на `Киселин` и `Основен` е да дадат определени поръчки на сервитьора, когато имат възможност, за да предадат достатъчно информация на `Зотеро` и той да може да определи техните индивидуални предпочитания. Понеже `Зотеро` не е учил химия и не може да разбере тези леки тънкости, достатъчно е да укаже или S, или $\{1, 2, \ldots, N\} \setminus S$.

Задача

Вашата задача е да измислите стратегия, която ще помогне на героите да се спасят от ресторанта, колкото се може по-бързо. Допълнително, понеже киселините и основите не са чак толкова различни, двамата клиенти **трябва да имат едно и също поведение** (т.е. трябва да имплементирате стратегия специално за ролята на главния готвач и още една стратегия специално за ролята на клиентите, независимо дали клиентът е `Киселин` или `Основен`).

Детайли по имплементацията

Това е задача тип комуникация. Грейдърът ще пусне три отделни инстанции на вашата програма, която трябва да имплементира както стратегията за главния готвач, така и за клиентите. Вашата програма не трябва да чете от стандартния вход или да отпечатва на стандартния изход. Трябва да включите хедъра `waiter.h` и да имплементирате следните функции, без да имплементирате функция `main`:

```
void initChef(int N);
std::vector<int> getOrder(int x);
void initCustomer(int N, int k, std::vector<int> S);
int makeOrder();
void waitTurn(int x);
```

Функцията `initChef` се извиква точно веднъж в инстанцията, съответстваща на главния готвач. Подава се N, броят на ястията в менюто. Функцията не трябва да връща нищо.

Функцията `getOrder` се извиква по веднъж за всяка поръчка, направена от някой клиент. Подава се x, цяло число, което е или -1 (задаващо мълчание), или е между 1 и N. Протоколът за стойността, която се връща е следната:

- Функцията трябва да върне `празен C++` вектор, ако главният готвач смята, че има нужда от още поръчки, преди да вземе решение.
- Функцията трябва да върне `непразен C++` вектор, когато главният готвач е определил предпочитанията, т.е. множеството S или $\{1, 2, \ldots, N\} \setminus S$, и съответно векторът съдържа елементите на съответното множество.

Функцията `initCustomer` се извиква точно веднъж във всяка инстанция, съответстваща на клиентите. Подава се N, броят на ястията в менюто; k, броят на ястията, предпочитани от текущия клиент; и S, `C++` вектор, съдържащ предпочитаните ястие на този клиент. Функцията не трябва да връща нищо.

Функцията `makeOrder` се извиква всеки път, когато поръчка трябва да се направи в точно една от инстанциите за клиентите. В другата инстанция за клиент ще има извикване на функцията `waitTurn` малко след това. На функцията `makeOrder` не се подават никакви стойности, но трябва да върне или -1 (в случай, че клиентът не иска да поръчва нищо), или стойност от множеството на клиента.

Функцията `waitTurn` се извиква след като е направена поръчка от инстанцията на другия клиент. Подава се x, което е поръчката, направена от другия клиент. Функцията не трябва да връща нищо.

Обърнете внимание, че двете инстанции, съответстващи на клиентите, не знаят кой процес съответства на `Киселин` и кой процес съответства на `Основен`.

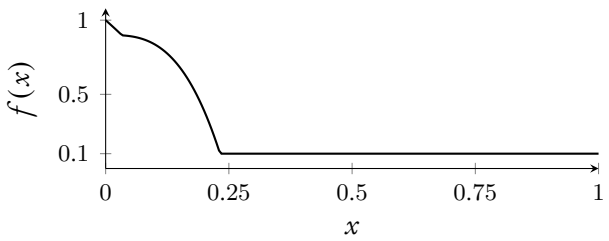
Грейдърът на журито е **адаптивен** от гледна точка на това в коя инстанция на клиент ще бъде извикана функцията `makeOrder` и в коя друга инстанция ще има извикване на функция `waitTurn`. Въпреки това **не е адаптивен** от гледна точка на избора на множеството S.

Оценяване

Вашето решение ще бъде оценено с 15 набора от тестове с подобни свойства. За всеки от тези набори ние определяме константата T_q като целевия брой поръчки за решаването на всеки тест от съответния набор. Нека C_q е най-големият брой поръчки, направени от вашето решение на някой тест от съответния набор. Ако вашето решение не мине теста или ако $C_q > 2 \cdot N$, няма да получите точки за този набор. Ако мине всички тестове на съответния набор и $C_q \leq T_q$, ще получите пълния брой точки за този набор.

За набор, решен с $T_q < C_q \leq 2 \cdot N$, нека $O = \frac{C_q}{T_q} - 1$ и нека дефинираме:

$$f(x) = \max\left(0.10, 1 - \min(0.10, 3x) - (4x)^3\right).$$



Резултатът ви на съответния набор ще е $P \cdot f(O)$, където P е определеният пълен резултат за този набор.

Имайки предвид, че N е броят на ястията в менюто, нека $K = \min(|S|, N - |S|)$. Тогава тестовете спазват следните ограничения:

Ограничения

#	Точки	Ограничения
0	0	Примерите от условието.
1	6	$N = 200\,000$, $K = 11$, $T_q = 1500$
2	4	$N = 200\,000$, $K = 111$, $T_q = 1500$
3	4	$N = 200\,000$, $K = 400$, $T_q = 1900$
4	7	$N = 200\,000$, $K = 700$, $T_q = 2100$
5	4	$N = 10\,000$, $K = 600$, $T_q = 700$
6	5	$N = 10\,000$, $K = 900$, $T_q = 700$
7	4	$N = 200\,000$, $K = 8000$, $T_q = 3350$
8	4	$N = 200\,000$, $K = 9000$, $T_q = 10000$
9	4	$N = 200\,000$, $K = 10000$, $T_q = 11250$
10	4	$N = 200\,000$, $K = 20000$, $T_q = 11350$
11	4	$N = 200\,000$, $K = 30000$, $T_q = 11350$
12	4	$N = 200\,000$, $K = 40000$, $T_q = 11400$
13	15	$N = 200\,000$, $K = 75000$, $T_q = 12900$
14	15	$N = 200\,000$, $K = 90000$, $T_q = 14600$
15	16	$N = 200\,000$, $K = 100000$, $T_q = 14650$

Примерен грейдър и инструмент за тестване

Предоставен ви е `testing_tool.py` заедно с `sample_grader.cpp`, за да ви помогнат да симулирате интеракцията на вашия компютър. Силно препоръчваме да ги използвате.

За да пуснете инструмента за тестване, добавете го в една папка заедно с `sample_grader.cpp` и вашата имплементация. Нека да предположим, че вашата програма се казва `sol.cpp`.

След това, трябва да отворите терминала в тази папка; за предоставените IDE-та (VS Code и CLion) това може да бъде направено директно от редактора, използвайки функционалността за помощ или на Linux, може да натиснете десен бутон в папката и да изберете "Open in Terminal". В този терминал напишете следната команда (и натиснете "Enter"):

```
python3 testing_tool.py sol.cpp test.txt
```

Обърнете внимание, че за тази команда, ако вашата програма има различно име, трябва да замените `sol.cpp` със съответното име.

В този случай ние също предполагаме, че входът за инструмента за тестване е от файл с име `test.txt`. Форматът на въпросния файл трябва да е следния:

- ред 1: N — брой ястия в менюто;
- ред 2: k — брой ястия, които клиент A предпочита;
- ред 3: $S_0 S_1 \ldots S_{k-1}$ — множеството от ястия, предпочитани от клиент A;
- ред 4: низ с произволна дължина, съдържащ A-та и B-та, които задават реда, в който сервитьорът трябва да приема поръчките на клиентите.

Може да откриете подробни инструкции за това как да използвате този инструмент за тестване в коментарите в началото на файла.

Предоставен ви е `sample_grader.cpp`, който може или да е в ролята на главния готвач, или на клиент. Той ще

извиква съответните функции от вашата имплементация.

За да изпълнявате примерния грейдър като инстанция на главния готвач, трябва да предоставите следните входни данни:

- ред 1: `chef` — низът "chef";
- ред 2: N — броят на ястията в менюто.

След това трябва да взаимодействате с него според отговорите, които получите от клиентите. За всяко такова взаимодействие, грейдърът извиква функцията `getOrder` и ако върне непразен вектор, той ще бъде отпечатан и програмата ще приключи.

За да изпълнявате примерния грейдър като инстанция на клиент, трябва да предоставите следните входни данни:

- ред 1: `customer` — низът "customer";
- ред 2: N k — броят на ястията в менюто и броят на предпочитаните ястия от клиента;
- ред 3: S_0 S_1 ... S_{k-1} — множеството от предпочитани ястия от клиента. Обърнете внимание, че трябва да пуснете две такива инстанции, които получават съответното си множество.

След това трябва да взаимодействате с него, както е описано в условието. Всеки път, докато главният готвач не отпечата отговор, трябва да изберете коя инстанция на клиент да питате за поръчка и след това да изпратите отговора към главния готвач и другия клиент. Когато питате клиент за неговата поръчка, трябва да въведете `!`. Когато го осведомявате за поръчка на другия клиент, трябва да въведете `.` , последвано от направената поръчка.

Примери

Нека приемем, че инструментът за тестване получава следния вход:

```
6
2
3 6
```

и че множеството $\{3, 6\}$ съответства на един от клиентите, нека го наречем A, а множеството допълнение $\{1, 2, 4, 5\}$ съответства на другия клиент, нека го наречем B.

Главният готвач		Клиент A		Клиент B	
вход	изход	вход	изход	вход	изход
`chef` 6		`customer` 6 2 3 6		`customer` 6 4 1 2 4 5	
  −1		 . −1		! 	 −1
  3		! 	 3 	 . 3	
  −1		 . −1		! 	 −1
  −1		 . −1		! 	 −1
  6	 3 6	! 	 6 	 . 6	

Накрая отговорът е правилен и инструмента за тестване ще отпечата такова съобщение.