Zadanie 1 (10 pkt. na pracowni, później 5 pkt.). W tym zadaniu celem jest napisanie bardzo uproszczonego modelu obługi finansów sklepu, który właśnie zaczyna swoją działalność.

Sklep oferuje MAX_PROD towarów (indeksowanych od 0 do MAX_PROD-1), ma INIT dukatów na koncie w ramach budżetu początkowego, a na towary nakłada MARGIN procent marży. Dukat jest jednostką walutową o dosyć niskiej wartości, jak jen czy forint, więc typowe ceny produktów są od setek dukatów wzwyż. Dukaty są najmniejszą jednostką w tym systemie (załóż, że wystarczą typy całkowitoliczbowe). Transakcje, które twój program powinien obsługować to:

- B [cena] [ilosc] [indeks] – jeśli sklep stać na zakup danej ilości towaru o danym indeksie po podanej cenie, sklep wykupuje ją, po czym należy odpowiednio zmodyfikować budżet sklepu i stan magazynu;
- S [ilosc] [indeks] – jeśli sklep ma odpowiednią ilość towaru na stanie, sprzedaje je; ponownie, budżet i stan magazynu należy zaktualizować;
- I – należy wypisać informację o stanie budżetu.

Twoje rozwiązanie powinno składać się z trzech plików: main.c, shop.h, shop.c. W pliku shop.h zdefiniuj:

- makra preprocesora MARGIN, INIT oraz MAX_PROD – odpowiednie stałe całkowite,
- typ złożony product zawierający pola price oraz amount. Cena produktu powina być zawsze ostatnią ceną, po jakiej sklep się w dany produkt zaopatrywał.

oraz zadeklaruj:

- statyczną zmienną całkowitoliczbową przechowującą informacje o budżecie sklepu,
- statyczną zmienną typu product *, pod którą w trakcie działania programu dostępna będzie tablica o MAX_PROD elementach,
- funkcje: init, buy, sell, info, które zdefiniujesz w pliku shop.c.

Funkcja init powinna być wywołana dokładnie raz, na początku funkcji main. Powinna ona alokować odpowiednią pamięć i inicjować obie ww. zmienne statyczne. W przypadku błędu alokacji pamięci, init powinna zwracać informację o błędzie – wtedy wypisz (w main!) odpowiedni komunikat i zakończ działanie programu. W funkcji sell budżet zmienia się o (100+MARGIN)/100 × cena produktu. Funkcje powinny mieć następujące sygnatury:

- int init();
- void buy(int index, int amount, int price);
- void sell(int index, int amount);
- void info();

Funkcja main w main.c powinna wywoływać init i w pętli wczytywać treść ze standardowego wejścia aż do EOF. Jeśli przeczytana sekwencja napisów jest poprawnym opisem transakcji (patrz wyżej), należy wywołać odpowiednią funkcję; w przeciwnym przypadku usunąć znak ze strumienia i próbować dalej.

Zadanie 2 (10 pkt.). To zadanie jest modyfikacją poprzedniego.

Pierwszą zmianą jest sposób przechowywania produktów: teraz zamiast indeksów będą rozróżniane po nazwach (składających się z małych liter alfabetu angielskiego). Pozbądź się makra MAX_PROD, zarządzaj potrzebną ilością pamięci przy użyciu malloc, calloc lub realloc. Użyj makra MAX_LNAME do przechowywania informacji o maksymalnej długości nazw produktów.

Dodatkowo produkty przechowuj w tablicy wskaźników. Przy zakupie nowego produktu (takiego, którego nazwa nie jest jeszcze obecna w tablicy) należy "dopisać" go do tablicy, z kolei w momencie odsprzedania całości zapasów któregoś produktu należy go usunąć, przenosząc na jego pozycję produkt z końca tablicy. W obu przypadkach należy dostosować wielkość tablicy – wskazanym sposobem jest przechowywanie kilku rekordów "zapasowych", niech ich jednak będzie mniej niż ok. ⅓ "prawdziwych". Pamiętaj o zwalnianiu nieużywanej pamięci.

W tej wersji może się zdarzyć błąd przy alokacji pamięci nie tylko w init, ale także w buy – ona też powinna

zwracać informację o ew. niepowodzeniu. W razie błędu wypisz odpowiedni komunikat z poziomu funkcji main.

Nowe sygnatury funkcji wyglądają następująco:

- int init();
- int buy(char * name, int amount, int price);
- void sell(char * name, int amount);
- void info();

Ostatnią zmianą będzie możliwość skompilowania programu z flagą -DDEBUG, która powinna sprawić, że także funkcje buy i sell (a nie tylko info) będą wypisywać informacje o przeprowadzonych transakcjach (i porażkach w ich przeprowadzeniu). Użyj w tym celu dyrektyw #ifdef itd.

Zadanie 3 (10 pkt.).

Dane jest *n* zbiorów liczb całkowitych, gdzie w *i*-tym z nich będą mogły się znajdować elementy od 0 do *m_i*-1. Twoim zadaniem jest obsłużenie następujących zapytań:

- + n d – dodaj do zbioru o numerze *n* wszystkie liczby z przedziału od 0 do *m_n*-1 podzielne przez *d* (takie, które już tam wcześniej były, po operacji dalej tam będą),
- - n d – usuń ze zbioru o numerze *n* wszystkie liczby z przedziału od 0 do *m_n*-1 podzielne przez *d* (takich, których tam wcześniej nie było, po operacji też tam nie będzie),
- ? n d – sprawdź czy w zbiorze o numerze *n* znajduje się wartość *d*.

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby *n* oraz *q*.

W drugim wierszu znajduje się *n* liczb: *m*_1, *m*_2, ... *m_n*.

W kolejnych *q* wierszach znajdują się opisy zapytań w formie przedstawionej powyżej.

Wyjście

Na wyjściu, dla każdego zapytania typu ? wypisz jedno słowo TAK lub NIE, w zależności czy dany element znajduje się w danym zbiorze, czy nie.

Ograniczenia

W tym zadaniu znowu kluczowe są limity:

- Liczba zbiorów $n \leq 1000$,
- Liczba zapytań $q \leq 10^5$,
- suma wartości *m_i* nie przekroczy 8×10^7,
- łączna liczba elementów potencjalnie dokładanych bądź wyjmowanych ze zbiorów nie przekroczy 5×10^7.

Uwaga: Limit pamięci znowu wynosi 14 MB.

Przykłady

Przykład A

Wejście

```
3 10
9 7 8
+ 0 2
? 0 6
? 0 3
+ 0 3
- 0 2
+ 1 2
? 0 6
? 0 3
? 1 6
? 2 6
```

Wyjście

TAK
NIE
NIE
TAK
TAK
NIE

Przykład B

Wejście

```
2 6
100 100
+ 0 1
+ 1 2
? 0 77
? 0 80
? 1 77
? 1 80
```

Wyjście

```
TAK
TAK
NIE
TAK
```

Przykład C

Wejście

```
5 12
260 215 13 11 2
+ 1 11
+ 0 71
+ 1 7
+ 4 1
- 1 19
- 0 44
+ 3 5
? 1 42
? 4 0
+ 2 7
? 3 9
? 0 88
```

Wyjście

```
TAK
TAK
NIE
NIE
```