Задача 7. Список с указателями

Источник:	основная*
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Удобно было бы реализовать связный список один раз, а потом использовать его в разных задачах для хранения разных типов данных (`int`, `double`, `char*`, ...). Однако сделать это не получается из-за того, что тип значения должен быть прописан в структуре узла. Самый простой способ решения этой проблемы — хранить в узлах списка не само значение, а указатель на него (т.е. его адрес). Если хранить внутри узла указатель `void*`, тогда в него можно будет записать указатель на значение любого типа.

В данной задаче нужно реализовать набор функций для работы со связным списком, в котором всегда хранятся указатели. Список должен быть двусвязный со вспомогательным узлом. В узлах списка требуется хранить указатели на следующий или предыдущей узлы (уже **не** индексы, как можно было в предыдущих задачах). Каждый узел списка должен быть размещён в динамической памяти, выделенной с помощью `malloc`.

Нужно реализовать следующие функции:

```
typedef struct Node_s {
    struct Node_s *prev, *next;
    void *value;
} Node;
//List -- вспомогательный узел, являющийся головой списка
typedef Node List;

//инициализирует поля структуры *list значениями для пустого списка
void initList(List *list);
//создаёт новый узел со значением ptr и вставляет его после узла node
//возвращает указатель на созданный узел
Node *addAfter(Node *node, void *ptr);
//создаёт новый узел со значением ptr и вставляет его перед узлом node
//возвращает указатель на созданный узел
Node *addBefore(Node *node, void *ptr);
//удаляет заданный узел, возвращая значение, которое в нём лежало
void *erase(Node *node);
```

Эти функции должны выделять/удалять память для узлов списка `Node`, но **не** для значений: вызывающий полностью контролирует то, куда указывают значения и как для них используется память.

Используя эти функции, решите тестовую задачу, аналогичную задаче “Двусвязный список”.

Формат входных данных

В первой строке файла записано одно целое число T — количество тестов в файле. Далее в файле идут тесты (T штук) подряд, один за другим.

Первая строка теста начинается с целого числа Q — количество операций, которые нужно выполнить ($0 \leqslant Q \leqslant 10^5$). В отличие от задачи “Двусвязный список”, в данной задаче полагается, что в начале теста список пустой.

Затем идут Q строк, которые описывают операции над списком. В каждой строке сначала записан тип операции: 1 — добавление спереди, -1 — добавление сзади, 0 — удаление. Затем указан индекс узла. Если описывается операция вставки, то в конце также задано целочисленное значение нового узла.

Узлам присваиваются индексы в порядке их создания. Самый первый созданный узел имеет индекс 0, следующая операция создания имеет индекс 1, и так далее. Индекс узла никогда не переиспользуется, даже после того, как узел удаляют из списка.

Все значения узлов лежат в диапазоне от 0 до 10^6 включительно.

Сумма Q по всем тестах не превышает 10^5.

Формат выходных данных

Для каждого теста нужно вывести значения всех узлов списка после выполнения операций (в порядке их следования в списке), и строку "===" в конце.

Пример

input.txt	output.txt
2 5 1 -1 4283 -1 0 2718 1 0 5000 -1 1 1111 1 1 3141 6 1 -1 0 1 -1 1000 -1 -1 2000 1 1 3000 -1 0 4000 0 2	1111 2718 3141 4283 5000 === 1000 3000 4000 0 ===

Пояснение к примеру

В примере два теста, похожие на тесты из задачи “Двусвязный список”. Различие лишь в том, что начальное состояние списка обеспечивается операциями, и третьего теста нет.