Задача 5. Снова растущий массив

Источник:	основная I
Имя входного файла:	`---`
Имя выходного файла:	`---`
Ограничение по времени:	8 секунд*
Ограничение по памяти:	разумное

В этой задаче предлагается реализовать псевдошаблонный код растущего массива, работающий для элементов любого типа. Этот код в дальнейшем можно легко подключить в любой задаче.

Вам нужно отправить два файла: `array_decl.h` и `array_def.h`. Файл `array_decl.h` будет многократно подключаться в хедере, чтобы сгенерировать объявление структуры и всех необходимых функций для работы с ней. Файл `array_def.h` будет многократно подключаться в файле исходного кода, чтобы сгенерировать определения всех функций, объявленных с помощью `array_decl.h`. Разрешается также отправить дополнительные хедеры.

Непосредственно перед каждым включением любого из ваших хедеров будет устанавлено два макроса `TYPE` и `NAME`. Первый задаёт известный на момент компиляции тип элемента, а второй определяет название массива (гарантируется, что он является корректным именем языка C). Например, чтобы объявить или определить массив под названием `array_long` с элементами типа `long`, перед включением будут установлены макросы:

```
#define TYPE long
#define NAME array_long
```

Разумеется, включения `array_decl.h` в хедере будут точно соответствовать включениям `array_decl.c` в файле с исходным кодом по количеству и типам и именам.

Растущий массив должен автоматически увеличивать размер буфера памяти по мере необходимости, так чтобы метод `push` работал за амортизированное время $O(1)$. В данной задаче запрещается автоматически уменьшать размер буфера памяти. Любой метод кроме `destroy` может лишь увеличить размер буфера, но не должен уменьшать его.

Ваши файлы должны объявлять/определять следующие вещи:

```
typedef struct array_long {
    int n;          //number of elements in array
    long *arr;      //points to the array of elements
    ...             //any other members can be added here
} array_long;

//initializes members of [vec] structure for empty array
void array_long_init(array_long *vec);

//makes array [vec] empty and frees its array buffer [vec->arr]
//note: this function can be called many times
void array_long_destroy(array_long *vec);

//adds element [value] to the end of array [vec]
//returns index of the added element
int array_long_push(array_long *vec, long value);

//removes the last element from array [vec]
//returns removed element
long array_long_pop(array_long *vec);

//ensures that array [vec] has enough storage for [capacity] elements
//note: address of elements surely won't change before [vec->n] exceeds capacity
void array_long_reserve(array_long *vec, int capacity);

//changes number of elements in array [vec] to [newCnt]
//if the number increases, new elements get value [fill]
//if the number decreases, some elements at the end are removed
void array_long_resize(array_long *vec, int newCnt, long fill);

//inserts elements [arr[0]], [arr[1]], [arr[2]], ..., [arr[num-1]]
//in-between elements [vec->arr[where-1]] and [vec->arr[where]]
//note: the whole array [arr] cannot be part of array [vec]
//[where] may vary from 0 to [vec->n], [num] can also be zero
void array_long_insert(array_long *vec, int where, long *arr, int num);

//removes elements [vec->arr[k]] for k = [where], [where+1], ..., [where+num-1]
//all the elements starting from [where+num]-th are shifted left by [num] positions
void array_long_erase(array_long *vec, int where, int num);
```

Указанные выше объявления проименованы для случая, когда `TYPE` имеет значение `long`, а имя `NAME` равно `array_long`. Если макросы имеют другое значение, имена структуры и функций также будут другими. Например, можно сделать массив указателей так:

```
#define TYPE void *
#define NAME array_pvoid
```

Тогда функция добавления элемента будет иметь сигнатуру:

```
int array_pvoid_push(array_pvoid *vec, void *value);
```

При тестировании ваш код будет включаться в тестовый код жюри согласно правилам. Ниже приведён пример того, как примерно будут использоваться ваши хедеры.

Хедер `sample.h`:

```
#pragma once

#define TYPE double
#define NAME vector
#include "array_decl.h"

#define TYPE int
#define NAME indices
#include "array_decl.h"
```

Файл исходного кода sample.c:

```c
#include "sample.h"
#include <assert.h>

#define TYPE double
#define NAME vector
#include "array_def.h"

#define TYPE int
#define NAME indices
#include "array_def.h"

int main() {
    vector values;
    vector_init(&values);
    assert(values.n == 0);
    vector_push(&values, 1.0);
    vector_push(&values, 2.0);
    assert(values.n == 2 && values.arr[0] == 1.0 && values.arr[1] == 2.0);
    indices ids;
    indices_init(&ids);
    int temp[] = {1,2,3,4,5,6};
    indices_insert(&ids, 0, temp, sizeof(temp)/sizeof(temp[0]));
    assert(ids.n == 6 && ids.arr[3] == 4);
    indices_insert(&ids, 4, temp, 5);
    assert(ids.n == 11 && ids.arr[3] == 4 && ids.arr[9] == 5);
    assert(ids.arr[4] == 1 && ids.arr[6] == 3 && ids.arr[8] == 5);
    indices_erase(&ids, 2, 5);
    assert(ids.n == 6 && ids.arr[0] == 1 && ids.arr[1] == 2);
    assert(ids.arr[2] == 4 && ids.arr[3] == 5 && ids.arr[4] == 5);
    indices_destroy(&ids); //memory freed
    indices_push(&ids, 13);
    assert(ids.n == 1 && ids.arr[0] == 13);
    indices_resize(&ids, 5, -1);
    assert(ids.n == 5 && ids.arr[0] == 13 && ids.arr[1] == -1);
    indices_reserve(&ids, 1000);
    int *ptr = &ids.arr[0];
    for (int i = 0; i < 900; i++) indices_push(&ids, i);
    assert(*ptr == 13 && ptr == &ids.arr[0]);
    for (int i = 0; i < 900; i++) indices_pop(&ids);
    assert(ptr == &ids.arr[0]); //never shrink buffer!
    indices_destroy(&ids);
}
```