Задача sm07-5: c/function-pointers/func-poliz-1

Напишите компилятор выражений в польской записи в "шитый" код, то есть в специальное внутреннее представление, удобное для исполнения, как описано далее.

Польская запись передается на вход в виде строки, в которой элементы польской записи отделяются друг от друга пробельными символами. Обозначим стек вычислений польской записи через S. На стеке вычислений хранятся 32-битные знаковые целые числа. Элемент S[0] — это верхушка стека, S[1] — это следующий после верхушки элемент на стеке. В польской записи поддерживаются следующие элементы:

`NUM`	NUM — это 32-битное целое число, перед которым допускается необязательные знаки "плюс" или "минус". значение числа помещается в стек.
`+`	Вычислить S[1] + S[0]. Оба значения удаляются из стека, результат операции помещается в стек.
`-`	Вычислить S[1] - S[0]. Оба значения удаляются из стека, результат операции помещается в стек.
`*`	Вычислить S[1] * S[0]. Оба значения удаляются из стека, результат операции помещается в стек.
`/`	Вычислить S[1] / S[0] (по математическим правилам нацело). Оба значения удаляются из стека, результат операции помещается в стек.
`%`	Вычислить S[1] % S[0] (по математическим правилам). Оба значения удаляются из стека, результат операции помещается в стек.
`#`	Вычислить -S[0]. Аргумент операции удаляется из стека, результат операции помещается в стек.
`r`	Считать со стандартного потока ввода 32-битное знаковое целое значение в десятичной записи, результат операции помещается в стек.
`w`	Вывести на стандартный поток вывода S[0]. Аргумент удаляется из стека.
`n`	Вывести на стандартный поток вывода символ \n.
`;`	Удалить элемент из верхушки стека.
`dNUM`	Поместить элемент S[NUM] на верхушку стека. Если NUM не указан, подразумевается значение индекса 0. Таким образом команда `d` копирует элемент на верхушке стека, как и команда `d0`. Команда `d1` заносит на верхушку стека значение S[1], где индекс берется до выполнения операции занесения в стек. Индекс NUM всегда неотрицательный.
`sNUM`	Обменять местами S[NUM] и S[0]. Если NUM не указан, подразумевается значение индекса 1. Таким образом команда `s` меняет местами S[1] и S[0], как и команда `s1`. Команда `s0` не делает ничего (даже если стек пуст). Индекс NUM всегда неотрицательный.

Предопределены следующие типы данных:

```
// opaque structure for poliz calculation state
struct PolizState;
```

```
// poliz operation handler
typedef int (*poliz_func_t)(struct PolizState *state, int iextra);

struct PolizItem
{
    poliz_func_t handler;
    int iextra;
};

// runtime errors
enum
{
    PE_OK,  // no error
    PE_STACK_UNDERFLOW, // not enough elements on stack
    PE_INVALID_INDEX, // s or d operations refer to invalid index
    PE_DIVISION_BY_ZERO,
    PE_INT_OVERFLOW,
    PE_READ_FAILED, // read from stdin failed to convert integer
for any reason
    PE_OUT_OF_MEMORY,
};

struct PolizItem *poliz_compile(const char *str);

struct PolizState *poliz_new_state(void);
void poliz_free_state(struct PolizState *state);
int poliz_last_error(struct PolizState *state);
```

Функция компиляции должна иметь следующий прототип:

```
struct PolizItem *poliz_compile(const char *str);
```

Функция компиляции возвращает массив элементов польской записи. Последний элемент массива содержит указатель handler равный NULL. Массив должен выделяться в динамической памяти.

Если дана строка `str`, то вычисление значения выполняется следующим образом:

```
    struct PolizItem *items = poliz_compile(str);
    struct PolizState *state = poliz_new_state();
    for (int i = 0; items[i].handler != NULL; ) {
        int err = items[i].handler(state, items[i].iextra);
        if (err < 0) {
            fprintf(stderr, "error: %d\n", -err);
            break;
        } else if (err > 0) {
            i = err;
        } else {
            ++i;
        }
    }
    poliz_free_state(state);
    free(items);
```

Ваша задача: написать функции `poliz_compile`, `poliz_new_state`, `poliz_free_state`, `poliz_last_error` и функции-обработчики команд польской записи. **Не сдавайте** код функции `main`. Вам будет доступен заголовочный файл `poliz.h`, который вы можете включать директивой #include.

Функция `poliz_last_error` возвращает код последней ошибки при выполнении польской записи. Если в процессе выполнения произошла ошибка, но выполнение не было прервано, то все последующие после ошибки команды ничего не делают, то есть в начале каждого обработчика команды должна находиться проверка:

```
    // проверяем была ли ошибка ранее
    if (state->err) return -state->err;
```

Используйте ключевое слово `static` там, где это полезно.

Можете предполагать, что польская запись корректна, за исключением возможных ошибок времени выполнения. То есть, польская запись может содержать ошибку деления на константу 0, но выявлять ее при компиляции, как и выявлять антипереполнение и другие ошибки времени выполнения не нужно.

Например, если дана строка `r r + w n`, при чтении со стандартного потока ввода

```
  100 128
```

на стандартный поток вывода должно быть напечатано

```
228
```

Обратите внимание, что длина вывода должна быть в точности 4 символа (3 цифры и \n).