P03 - калькулятор (Козловская А., группа 207)

P03 состоит из модулей main.c, poliz.c, stack.c, calc.c, init_var.c, recurs_expl.c, соответствующих заголовочных файлов и заголовочного файла constants.h.
Программа работает с числами типов long long int и double.
На вход подаётся математическое выражение, состоящее из переменных (длина имени не превышает 6 символов, имя состоит из латинскиз букв, цифр и знака '_'), констант типов long long int и double, знаков математических операций (+, -, *, /), круглых скобок.
Признаком конца математического выражения является перевод строки или EOF.
Программа считывает выражение, и, если оно корректно, принимает на вход выражения вида <имя_переменной> = <значение_переменной>. Признаком конца ввода является конец файла.
Далее программа выводит результат(stdin) или сообщение с описанием ошибки (stderr).

==========

Возможные ошибки:
1) Нарушен баланс скобок (выводится, каких скобок больше)
2) Отсутствие операнда
3) Пропуск операции
4) Деление на ноль
5) Неверный операнд
6) Неинициализированная константа
7) Пустой стек (при отсутствии операндов)
8) Ошибка выделения памяти

==========

constants.h

Описанные типы и констаны:

1) константы возможных типов элементов выражения (целочисленная константа, вещественная константа, переменная, операции +, -, *, / соответственно), а так же константу, означающую конец выражения.

```
enum expr_item_types
{
    eit_int,  eit_dbl,   eit_var,
    eit_plus, eit_minus, eit_mul,
    eit_div,  eit_lp,    eit_pp,
    eit_end
};
```

2) Структура, содержащая один из элементов выражения (см. expr_item_types)

```
typedef struct Expression_item
{
    int type;
    union {
        long long i;
        double d;
        char var[sizeof(double)];
    } data;
} Expression_item;
```

2) Структура, содержащая одну из переменных (её имя str, тип значения type и значение data.i или data.d для целочисленных или вещественных переменных соответственно)

```
typedef struct Variable_value
{
    char str[sizeof(double)];
    int type;
    union {
        long long i;
        double d;
    } data;
} Variable_value;
```

3) Структура, содержащая таблицу переменных.

```
typedef struct Var_table
{
    size_t size;
    Variable_value *data;

} Var_table;
```

4) Константы, задающие коды ошибок.

```
enum Errors
{
    STACK_INIT_ERR = 1,   MEM_ERR,    NO_CONST_ERR,
    EMPTY_STACK_ERR,      LP_ERR,     DIV_ZERO_ERR,
    DATA_SIZE_ERR,        PP_ERR,     LITER_ERR,
    VAR_NOT_FOUND_ERR,    MIS_OPRND,  MIS_OPRTN
};

enum {VAR_NUM = 5};
```

==========

==========

stack.c и stack.h

Описанный тип:

```
typedef struct Stack
{
    void *data;
    size_t size;

} Stack;
```

С помощью модуля stack.c осуществляются действия со стеком:

1) Функция init_stack инициализирует стек.

```
int init_stack(Stack *stack);
```

В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

2) Функция push_stack записывает данные в стек.

```
int push_stack (Stack *stack, const void *data, size_t size_data);
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

3) Функция pop_stack извлекает данные из стека.

```
int pop_stack(Stack *stack, const void *resp, size_t size_res);
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

4) Функция final_stack очищает стек.

```
void final_stack(Stack *stack);
```

==========
==========

poliz.c и poliz.h

Описанные типы:

```
typedef struct Poliz
{
      void *data;
         size_t cap;   // what size of memory is allocated
      size_t size;  // how much memory is actually occupied
} Poliz;


typedef char Size_elem;

typedef int (*Calculate_elem)(const void *elem, Size_elem size, Stack
*stack);
```

С помощью модуля poliz.c осуществляются действия с обратной польской записью:

1) Функция calculate_poliz вычисляет полиз.

```
int calculate_poliz(const Poliz *pol, void *resp, size_t size_res)
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

2) Функция init_poliz инициализирует полиз.

```
int init_poliz(Poliz *poliz)
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

3) Функция deinit_poliz деинициализирует полиз.

```
int deinit_poliz(Poliz *poliz)
```
В любом случае возвращает 0.

4) Функция new_elem_poliz добавляет новый элемент в полиз.

```
int new_elem_poliz(Poliz *poliz, void *item, Size_elem size,
Calculate_elem *func)
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

==========
==========

recurs_expl.c и recurs_expl.h

Описанный тип:

```
typedef struct Help_struct
{
    int buf;
    int operand;
    int num_of_lp;
    int num_of_pp;

} Help_struct;
```

Вспомогательная структура для контроля ввода.

С помощью модуля recurs_expl.c осуществляется разбор входной строки методом рекурсивного спуска:
Следующие функции реализуют данный метод и в случае ошибки возвращают код ошибки, при корректной работе возвращают 0.

```
int rec_expl_next_char(Expression_item *cur_expr, Help_struct *ctrl);
int rec_expl_add_sub(Poliz *poliz, Expression_item *curc, Help_struct
*ctrl);
int rec_expl_mul_div(Poliz *poliz, Expression_item *curc, Help_struct
*ctrl);
int rec_expl_symbol(Poliz *poliz, Expression_item *curc, Help_struct
*ctrl);
```

==========
==========

init_var.c и init_var.c

С помощью данного модуля происходит считывание(с контролем ввода) значений переменных.
1) Функция get_var считывает каждую новую переменную и её значение.

```
int get_var(Variable_value *cur_var);
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

2) Функция get_var_tabl формирует так называемую "таблицу переменных" variables, с помощью которой очуществляется замена переменной её значением при вычислениях.

```
int get_var_tabl(Var_table *my_table);
```
В случае ошибки возвращает код ошибки, при корректной работе возвращает 0.

==========
==========
calc.c и calc.h

Модуль calc.c содержит функции для вычисления обратной польской записи:

```
int calc_const(const void *elem, Size_elem size, Stack *stack);
int calc_var(const void *elem, Size_elem size, Stack *stack);
int calc_plus(const void *elem, Size_elem size, Stack *stack);
int calc_minus(const void *elem, Size_elem size, Stack *stack);
int calc_mul(const void *elem, Size_elem size, Stack *stack);
int calc_div(const void *elem, Size_elem size, Stack *stack);
```

для констант, переменных, операций сложения, вычитания, умножения и деления соответственно.
В случае ошибки возвращают код ошибки, при корректной работе возвращают 0.
==========