Лабораторная работа №2
по дисциплине «Типы и структуры данных»
Тема :Работа со стеком.

Условие задачи:

Разработать программу работы со стеком, реализующую операции добавления и удаления элементов из стека и отображения текущего состояния стека. Реализовать стек: а) массивом; б) списком.

Проверить правильность расстановки скобок трех типов (круглых, квадратных и фигурных) в выражении.

Реализация стандартных операций со стеком

Выбор пользователем метода реализации стека:

```
Стек массивом(1) или стек списком(2):
```

Ввод пользователя: 1 - реализация стека массивом
2 - реализация стека списком

Выбор действия со стеком:

```
Push(1) Pop(2) Print(3) Info(4) Exit(0):
```

Ввод пользователя: 1 - добавить новый элемент в стек
2 - удалить элемент из стека
3 - напечатать элементы стека (и их адреса)
4 - вывести на экран затраченную память и в реализации списком область свободных областей
0 - завершение программы

Выполнение задания лабораторной работы

Исходные данные:

выражение (вводится с клавиатуры), содержащее знаки математических операций, числа и скобки трех видов: круглые, фигурные и квадратные (не обязательно все)

Ввод:

корректный:

- { 1 * [1 + 1] / 1 } * (1 + 1)
- [( )]
- [1 +] * ( )
- (1 + (1 + (1 + 1)))
- } { 1 + } (

некорректный:

- abc
- [a + b] * ( )
- abc)

Вывод:

В качестве результата программа печатает:

- Ошибка в расстановке скобок в выражении
  или
- Скобки в выражении расставлены верно

Возможные ошибки пользователя:

Некорректный ввод выражения

Реакция - сообщение “Неверный ввод” и завершение программы

Превышение длины выражения

Реакция - сообщение “В выражении должно быть не более N символов” и завершение программы

Тесты:

{) (( ]	Ошибка в расстановке скобок в выражении
{ } (({ }))	Скобки в выражении расставлены верно

Алгоритм:

1. Ввод выражения
2. Проверка на корректность ввода
3. Заполнение стека в виде массива элементами
4. Печать полученного стека
5. Проверка правильности расстановки скобок
6. Печать результата
7. Печать дополнительной информации (затраченное время, используемая память)
8. Заполнение стека в виде списка элементами
9. Печать полученного стека
10. Проверка правильности расстановки скобок
11. Печать результата
12. Печать дополнительной информации (затраченное время, используемая память, свободные области памяти)
13. Освобождение памяти, занимаемой списком

Структура элемента стека:

Реализация списком	**Реализация массивом**
struct stack_tl **{** **char data;** **struct stack_tl *next;** **};** data - информация next - ссылка на следующий элемент стека	**struct stack_ta** **{** **char data[N];** **int size;** **};** data - информация size - размер стека

Общие функции:

- *char opposite_bracket(char bracket)*

назначение: возвращает противоположный символ скобки
параметры: bracket символ скобки
возвращает: символ скобки, противоположной bracket

- *void free_all(struct stack_tl **head)*

назначение: освобождение памяти, занимаемой списком
параметры head стек в виде списка

- *int size_l(struct stack_tl *head)*

назначение:считает количество элементов стека
параметры: head стек в виде списка
возвращает: количество элементов стека

Функции для реализации стека массивом и списком:

- *void push_a(struct stack_ta *stack, char value)*
- *void push_l(struct stack_tl **head, char data)*

назначение: добавляет элемент в стек
параметры: stack / head стек
value / data данные, помещаемые в стек

- *int pop_a(struct stack_ta *stack)*
- *struct stack_tl* pop_l(struct stack_tl **head)*

назначение: удаляет элемент из стека
параметры: stack / head стек
возвращает: код ошибки / указатель на голову стека

- *void print_stack_a(struct stack_ta *stack)*
- *void print_stack_l(struct stack_tl *head)*

назначение: выводит на экран элементы стека и их адреса
параметры: stack / head стек

- *int filling_stack_a(struct stack_ta *stack, const char *string, int *size)*
- *int filling_stack_l(struct stack_tl **head, const char *string)*

назначение: заполняет стек элементами, проверяет некорректность вводимых данных

параметры stack / head стек
string введенное выражение
size размер стека
возвращает: код ошибки (некорректные данные или ок)

- *int task_a(struct stack_ta *stack)*
- *void task_l(struct stack_tl *head, char **array_free)*

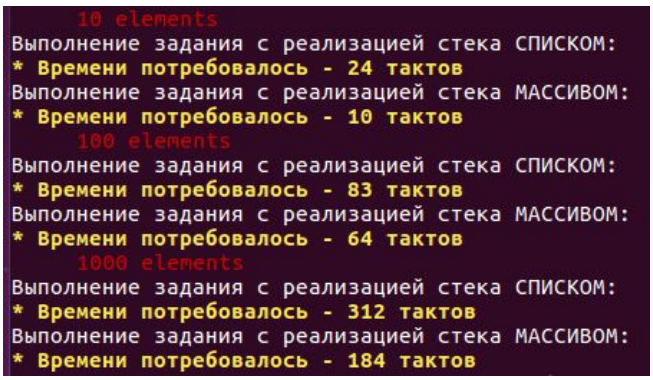
назначение: проверяет правильность расстановки скобок в выражении
параметры: stack стек
возвращает: код ошибки (верно, неверно, некорректные данные)
алгоритм:

```
если stack.size <> делать
        если верхний элемент stack == “{“ или “(“ или “[”
                error = OKFAIL
        иначе
                пока stack.size > 0
                        если new_stack.size == 0
                                push_a(new_stack, stack.data[stack.size-1])
                        иначе если stack.data[stack.size-1] <>
                        opposite_bracket(new_stack.data[new_stack.size-1])
                                push_a(new_stack, stack.data[stack.size-1])
                        иначе
                                pop_a(new_stack)
                        pop_a(stack)
                если new_stack.size == 0
                        error = OKPASS
                иначе
                        error = OKFAIL
иначе
        error = INCORRECT
```

Реализация стека списком и массивом

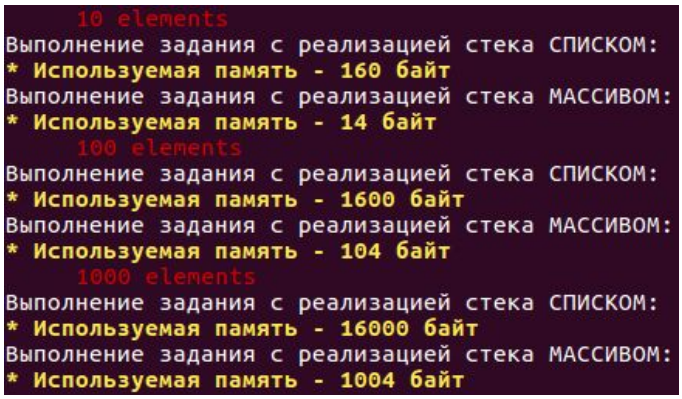
Время(мс):

	Список	*Массив*	*%*
10 элементов	*24*	*10*	*240*
100 элементов	*83*	*64*	*130*
1000 элементов	*312*	*184*	*170*



Память (байт):

	Список	*Массив*
10 элементов	*160*	*14*
100 элементов	*1600*	*104*
1000 элементов	*16000*	*1004*



***Вывод**: реализация стека массивом эффективнее по времени в 1.5-2 раза и эффективнее по памяти примерно в 16 раз, так как при реализации стека списком большое количество памяти отводится на хранение указателей. Таким образом, на один элемент списка требуется 16 байт, а на хранение элемента массива нужен 1 байт и еще 4 байта на хранение размера стека.*

Вопросы к лабораторной работе

1. **Что такое стек?**

Стек – это последовательный список с переменной длиной, в котором включение и исключение элементов происходит только с одной стороны – с его вершины.

2. **Каким образом и сколько памяти выделяется под хранение стека при различной его реализации?**

Если стек реализован в виде массива, то для его хранения обычно отводится непрерывная область памяти ограниченного размера.
При реализации стека списком память, доступная для хранения элементов ограничена только объемом свободной RAM. Память выделяется для каждого элемента при его добавлении.

3. **Каким образом освобождается память при удалении элемента стека при различной реализации стека?**

При реализации списком память освобождается при удалении элемента, который ее занимал. При реализации массивом память освобождается в конце работы со стеком: освобождается весь массив.

4. **Что происходит с элементами стека при его просмотре?**

Операция чтения производится с помощью операции извлечения элемента из стека.

5. **Каким образом эффективнее реализовывать стек? От чего это зависит?**

Эффективность определяется типом и количеством хранимых данных. Например, при хранении в стеке-списке элементов типа char или int, память используется неэффективно, так как на хранение указателей на другие элементы выделяется больше, чем на значение элемента. Также, при реализации списком память выделяется в разных областях оперативной памяти, что не позволит эффективно использовать предоставленную программе память.