Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Работа со стеком

Вариант 6

Студент **Ильченко Е. А.**

Группа **ИУ7-34Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Ильченко Е. А.**

Преподаватель _____ **Никульшина Т. А.**

2024 г.

Описание условия задачи

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек:
а) статическим массивом (дополнительно можно реализовать динамическим массивом);
б) списком.
Все стандартные операции со стеком должны быть оформлены подпрограммами. При реализации стека списком в вывод текущего состояния стека добавить просмотр адресов элементов стека и создать свой список или массив свободных областей (адресов освобождаемых элементов) с выводом его на экран.
Используя стек, определить, является ли строка палиндромом

Описание ТЗ

1. Описание исходных данных и результатов работы программы

Входные данные:
Пользовательская команда из доступных, содержимое файла и необходимые аргументы определенного сценария:
0 – Выйти

Операции со стеком, реализованным как массив
1 – Добавить символ в стек
2 – Удалить символ из стека
3 – Вывести стек с адресами элементов
4 – Посмотреть вершину стека
5 – Очистить стек

Операции со стеком, реализованным как список
6 – Добавить символ в стек
7 – Удалить символ из стека
8 – Вывести стек с адресами элементов
9 – Посмотреть вершину стека
10 – Очистить стек

11 – Вывести меню
12 – Определить является ли строка палиндромом

13 – Замерить время работы разных стеков

Выходные данные:
Стек, измененный в соответствии с выбранной операцией

2. Описание задачи, реализуемой в программе

Задача, реализуемая в программе, заключается в реализации операции работы со стеком, который представлен в виде статического массива и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации; получить представление о механизмах выделения и освобождения памяти при работе со стеком.

3. Способ обращения к программе

Запуск исполняемого файла

```
./app.exe
```

Далее выбирается, какой пункт меню выполнить

4. Описание возможных аварийных ситуаций и ошибок пользователя

Список аварийных ситуаций:

- Ошибка ввода данных `IO_ERROR`
- Ошибка ввода команды `COMMAND_ERROR`

5. Описание внутренних структур данных

Структура данных стека, как массива:

```
#define STACK_CAPACITY 1000

typedef struct {
 char array[STACK_CAPACITY];
 int top;
 int capacity;
} stack_array_t;
```

Структура данных стека, как списка

```
typedef struct
{
   char data;
```

```
   struct node_t *next;
} node_t;

typedef struct
{
   node_t *top;
} stack_list_t;
```

6. Описание функций

Функции для работы со стеком, как массивом

```
void init_stack_arr(stack_array_t *stack);
```

Функция инициализации стека

```
int push_arr(stack_array_t *stack, char element);
```

Функция добавления элемента в стек

```
char pop_arr(stack_array_t *stack);
```

Функция удаления элемента из стека

```
void print_arr(stack_array_t *stack);
```

Функция вывода стека

```
void peek_arr(stack_array_t *stack);
```

Функция вывода верхнего элемента стека

```
void free_stack_arr(stack_array_t *stack);
```

Функция очищения стека

```
int is_palindrome_arr(stack_array_t *stack);
```

Функция проверки строки на палиндромность

Функции для работы со стеком, как списком

```
void init_stack_list(stack_list_t *stack);
```

Функция инициализации стека

```
int push_list(stack_list_t *stack, char element);
```

Функция добавления элемента в стек

```
char pop_list(stack_list_t *stack);
```

Функция удаления элемента из стека

```
void print_list(stack_list_t *stack);
```

Функция вывода стека

```
void peek_list(stack_list_t *stack);
```

Функция вывода верхнего элемента стека

```
void free_stack_list(stack_list_t *stack);
```

Функция очищения стека

```
int is_palindrome_list(stack_list_t stack);
```

Функция проверки строки на палиндромность

Функции для замера времени

```
void measure_time(void);
```

Функция для замера времени добавления и удаления элементов в стеки

7. Описание алгоритма

Алгоритмы для стека, как массива

Алгоритм вставки элемента

1. Проверить, заполнен ли стек, вызвав `is_full_arr(stack)`. Если заполнен, вернуть 1.
2. Увеличить указатель `top`, чтобы указать на следующую свободную ячейку.
3. Записать `element` в `stack->array[stack->top]`.
4. Вернуть 0 для успешного завершения операции.

Алгоритм удаления элемента

1. Проверить, пуст ли стек, вызвав `is_empty_arr(stack)`. Если стек пуст, вернуть символ '\0' как признак ошибки.
2. Сохранить верхний элемент стека `stack->array[stack->top]` во временную переменную `value`.
3. Уменьшить указатель `top`, чтобы переместить его на элемент ниже.

4. Вернуть сохранённое значение `value`.

Алгоритмы для стека, как списка

Алгоритм вставки элемента

1. Выделить память для нового узла `node` с помощью `malloc`. Если выделение памяти не удалось, вернуть 1.
2. Присвоить полю `data` нового узла значение `element`.
3. Установить указатель `next` нового узла на текущий верхний элемент `stack->top`.
4. Обновить указатель `top` стека так, чтобы он указывал на новый узел.
5. Вернуть 0, чтобы обозначить успешное добавление элемента в стек.

Алгоритм удаления элемента

1. Проверить, пуст ли стек, вызвав `is_empty(stack)`. Если стек пуст, вернуть '\0'.
2. Сохранить указатель на текущий верхний элемент `stack->top` во временную переменную `temp`.
3. Сохранить данные текущего верхнего элемента `stack->top->data` в переменную `value`.
4. Обновить `stack->top`, чтобы он указывал на следующий элемент `stack->top->next`.
5. Освободить память, занятую временным узлом `temp`.
6. Вернуть значение `value`.

Алгоритм проверки строки на палиндром

1. Проверка на пустой стек. Если стек пуст (`is_empty_arr`), сразу возвращается 1, так как пустая строка считается палиндромом.
2. Инициализация вспомогательного стека. Создается новый стек `stack_reversed` с такой же емкостью, как у исходного стека. Вызывается функция `init_stack_arr`, чтобы инициализировать этот стек.
3. Копирование элементов в перевернутый стек.
    a. Проходимся по индексам элементов в исходном стеке (от 0 до `stack->top`).
    b. Для каждого элемента вызывается `push_arr`, чтобы вставить элемент в `stack_reversed`.
    c. Если при вставке происходит ошибка (например, нехватка памяти), освобождается память для `stack_reversed`, и функция возвращает 0.
4. Сравнение элементов исходного и перевернутого стеков.

a. Создается указатель `p`, который начинает с первого элемента исходного массива `stack->array`.
b. Пока перевернутый стек `stack_reversed` не пуст:
    i. Извлекается верхний элемент `stack_reversed` с помощью `pop_arr`.
    ii. Сравнивается извлеченный элемент с текущим элементом исходного стека.
    iii. Если данные не совпадают, освобождается память для `stack_reversed`, и функция возвращает 0.
5. Очистка памяти. После завершения всех сравнений освобождается память, выделенная для `stack_reversed`.
6. Если все элементы совпали, возвращается 1, что означает, что строка в стеке является палиндромом.

Тесты

Тесты для стека на основе массива

Тест	**Описание теста**	**Шаги теста**	**Ожидаемый результат**
1	Проверка инициализации стека	1. Создать стек. 2. Вызвать init_stack_arr.	Стек должен быть инициализирован, top равен -1.
2	Проверка вставки одного элемента	1. Создать стек и инициализировать его. 2. Вызвать push_arr с элементом 'a'.	Стек содержит один элемент: ['a'], top равен 0.
3	Проверка вставки нескольких элементов	1. Создать стек и инициализировать его. 2. Вызвать push_arr с элементами 'a', 'b', 'c'.	Стек содержит элементы: ['a', 'b', 'c'], top равен 2.
4	Проверка переполнения стека	1. Создать стек с фиксированной емкостью (например, 3). 2. Заполнить стек 3 элементами: 'a', 'b', 'c'.	Возвращает 1 (ошибка переполнения).

		3. Попробовать добавить 4-й элемент 'd'.	
5	Проверка извлечения элемента	1. Создать стек и инициализировать его. 2. Добавить элементы: 'a', 'b', 'c'. 3. Вызвать pop_arr.	Возвращает 'c', стек теперь содержит ['a', 'b'], top равен 1.
6	Проверка извлечения из пустого стека	1. Создать стек и инициализировать его. 2. Вызвать pop_arr.	Возвращает '\0' (стек пуст).
7	Проверка на палиндром	1. Создать стек и инициализировать его. 2. Добавить элементы ['a', 'b', 'a']. 3. Вызвать is_palindrome_arr.	Возвращает 1 (строка является палиндромом).
8	Проверка на не палиндром	1. Создать стек и инициализировать его. 2. Добавить элементы ['a', 'b', 'c']. 3. Вызвать is_palindrome_arr.	Возвращает 0 (строка не является палиндромом).
9	Проверка на пустоту	1. Создать стек и инициализировать его. 2. Вызвать is_empty_arr.	Возвращает 1 (стек пуст).
10	Проверка на непустой стек	1. Создать стек и инициализировать его. 2. Добавить элемент 'x'. 3. Вызвать is_empty_arr.	Возвращает 0 (стек не пуст).
11	Печать элементов стека	1. Создать стек и инициализировать его. 2. Добавить элементы 'x', 'y', 'z'. 3. Вызвать print_arr.	Печатает адреса и элементы: z, y, x.

12	Просмотр верхнего элемента	1. Создать стек и инициализировать его. 2. Добавить элементы 'x', 'y', 'z'. 3. Вызвать peek_arr.	Печатает адрес и значение верхнего элемента: z.
13	Просмотр верхнего элемента из пустого стека	1. Создать стек и инициализировать его. 2. Вызвать peek_arr.	Печатает "Стек пустой".

Тесты для стека на основе связанных списков

Тест	**Описание теста**	**Шаги теста**	**Ожидаемый результат**
1	Проверка инициализации стека	1. Создать стек. 2. Вызвать init_stack_list.	Стек должен быть инициализирован, top равен NULL.
2	Проверка вставки одного элемента	1. Создать стек и инициализировать его. 2. Вызвать push_list с элементом 'a'.	Стек содержит один элемент: ['a'], top указывает на 'a'.
3	Проверка вставки нескольких элементов	1. Создать стек и инициализировать его. 2. Вызвать push_list с элементами 'a', 'b', 'c'.	Стек содержит элементы: ['c', 'b', 'a'], top указывает на 'c'.
4	Проверка извлечения элемента	1. Создать стек и инициализировать его. 2. Добавить элементы: 'a', 'b', 'c'. 3. Вызвать pop_list.	Возвращает 'c', стек теперь содержит ['b', 'a'], top указывает на 'b'.
5	Проверка извлечения из пустого стека	1. Создать стек и инициализировать его. 2. Вызвать pop_list.	Возвращает '\0' (стек пуст).
6	Проверка на палиндром	1. Создать стек и инициализировать его. 2. Добавить элементы ['a', 'b', 'a'].	Возвращает 1 (строка является палиндромом).

		3. Вызвать is_palindrome_list.	
7	Проверка на не палиндром	1. Создать стек и инициализировать его. 2. Добавить элементы ['a', 'b', 'c']. 3. Вызвать is_palindrome_list.	Возвращает 0 (строка не является палиндромом).
8	Проверка на пустоту	1. Создать стек и инициализировать его. 2. Вызвать is_empty.	Возвращает 1 (стек пуст).
9	Проверка на непустой стек	1. Создать стек и инициализировать его. 2. Добавить элемент 'x'. 3. Вызвать is_empty.	Возвращает 0 (стек не пуст).
10	Печать элементов стека	1. Создать стек и инициализировать его. 2. Добавить элементы 'x', 'y', 'z'. 3. Вызвать print_list.	Печатает адреса и элементы: z, y, x.
11	Просмотр верхнего элемента	1. Создать стек и инициализировать его. 2. Добавить элементы 'x', 'y', 'z'. 3. Вызвать peek_list.	Печатает адрес и значение верхнего элемента: z.
12	Просмотр верхнего элемента из пустого стека	1. Создать стек и инициализировать его. 2. Вызвать peek_list.	Печатает "Стек пустой".
13	Освобождение памяти стека	1. Создать стек и инициализировать его. 2. Добавить элементы 'a', 'b', 'c'. 3. Вызвать free_stack_list.	Все элементы удалены, стек пуст, top равен NULL.

Оценка эффективности

Эффективность по времени/памяти считалась путем замера 100 раз методов удаления/вставки одного элемента разных реализаций стека и усреднения результатов
Выгода считалась, как
$\frac{StackList - StackArray}{StackArray} \cdot 100$
Измерения проводились на MacBook Pro 13 2019

Стек заполнен на **5%** от размера стека как массива

Стек как массив			**Стек как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление	Вставка	Память
1.23	1.65	1008	3.14	2.86	458	155.3	73.3	-54.6

Стек заполнен на **7%** от размера стека как массива

Стек как массив			**Стек как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление	Вставка	Память
1.34	1.74	1008	4.06	3.58	638	203.0	105.7	-36.7

Стек заполнен на **10%** от размера стека как массива

Стек как массив			**Стек как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Удаление, сек$^{-6}$	Вставка,	Память, байт	Удаление,	Вставка, сек$^{-6}$	Память, байт	Удаление	Вставка	Память

	сек$^{-6}$		сек$^{-6}$					
1.70	1.98	1008	5.29	4.59	908	211.2	131.8	-9.9

Стек заполнен на **50%** от размера стека как массива

Стек как массив			**Стек как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление	Вставка	Память
3.40	4.12	1008	22.91	19.51	4508	573.8	373.5	347.2

Стек заполнен на **100%** от размера стека как массива

Стек как массив			**Стек как список**			**Выгода по времени, %**		**Выгода по памяти, %**
Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление, сек$^{-6}$	Вставка, сек$^{-6}$	Память, байт	Удаление	Вставка	Память
5.89	6.34	1008	45.23	38.31	9008	667.9	504.3	793.7

Вывод

Выполнение данной лабораторной работы позволяет сделать следующие выводы

1. Если мы хотим реализовать более быстро работающий стек, то мы должны использовать стек на основе массива. Но мы столкнемся с ограничениями по памяти:
    a. В памяти недостаточно места для выделения под большой массив
    b. Память фрагментирована, поэтому она не сможет выделить непрерывный кусок памяти для массива
2. Если мы хотим реализовать динамически расширяемый стек, то мы должны использовать стек на основе списка. Но мы столкнемся с более медленной работой методов стека

Ответы на контрольные вопросы

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. Стек функционирует по принципу: последним пришел – первым ушел, Last In – First Out (LIFO).

2. **Каким образом и сколько памяти выделяется под хранение стека при различной его реализации?**
   При реализации стека, как статического массива:
   Выделяется непрерывная область памяти ограниченного размера сразу на все элементы стека, имеющая нижнюю и верхнюю границу. Также заводится число, указывающее, какой элемент является верхним в стеке и заводится число, хранящее размер стека.

   При реализации стека, как списка:
   До начала работы указатель стека показывает на нулевой адрес. При включении элемента в стек сначала происходит выделение области памяти, адрес которой записывается в указатель стека, а затем по значению этого указателя в стек помещается информация.

3. **Каким образом освобождается память при удалении элемента стека при различной реализации стека?**
   При реализации стека, как статического массива:
   При исключении элемента считывается информация об исключаемом элементе, а затем уменьшается на единицу число, хранящее индекс верхнего элемента. Память освобождается при окончании работы по стеком целиком

   При реализации стека, как списка:
   При исключении элемента сначала по указателю стека считывается информация об исключаемом элементе, а затем указатель смещается к предыдущему элементу. После чего освобождается память, выделенная под элемент

4. **Что происходит с элементами стека при его просмотре?**
   При просмотре элементов стека они удаляются из него

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

Стек на статическом массиве эффективнее стека на связанном списке, так как:

1. Быстрее доступ: элементы хранятся в непрерывной памяти.
2. Экономия памяти: не нужны дополнительные указатели.
3. Быстрее операции: push и pop не требуют выделения памяти.
4. Нет динамического выделения: память выделяется один раз при инициализации.

Но если нужно:

1. Динамическое изменение размера
2. Мы не знаем заранее, сколько памяти нам нужно
3. Наш стек очень большой и мы не уверены, что в памяти сможет выделиться непрерывный кусок памяти для стека

То лучше использовать стек на связанном списке.