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»

Отчёт

по лабораторной работе №4

Название «Работа со стеком»

Дисциплина «Типы и структуры данных»

Вариант 2

Студент ИУ7-31Б _____ Корниенко К.Ю.
(подпись, дата) (Фамилия И.О.)

Преподователь _____ Никульшина Т.А.
(подпись, дата) (Фамилия И.О.)

Москва, 2021

Содержание

Введение

Стек — одна из наиболее часто используемых структур данных в программировании. Данная СД является незаменимой в современном программировании. Стеки используются для организации вызова подпрограмм, адресов возврата.

Целью данной лабораторной работы является реализация операций работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного списка, оценить преимущества и недостатки каждой реализации, получить представление о механизмах выделения и освобождения памяти при работе с динамическими структурами данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) Исследовать варианты реализации стеков.
2) Привести схемы алгоритмов вставки элемента в стек и удаления из стека.
3) Описать используемые структуры данных.
4) Оценить объем памяти для хранения данных.
5) Описать требования к ПО.
6) Протестировать разработанное ПО.
7) Сравнить реализации стеков на списке и массиве.

1 Аналитический раздел

В данном разделе будет представлено описание предметной области и описаны особенности реализации стеков на списке и на массиве. Также будет описано условие и техническое задание лабораторной работы.

1.1 Сведения о стеке

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)**.

Программная реализация стека возможна на основе различных структур данных, например, с использованием статических или динамических одномерных массивов (векторов) и линейных списков.

При работе со стеком доступен только его верхний элемент, который адресуется специальным указателем стека PS (Pointer Stack). Причем классическая реализация стека предполагает, что просмотреть содержимое стека без извлечения (удаления) его элементов невозможно.

При работе со стеком необходимо отслеживать возникновение таких аварийных ситуаций, как попытки исключения элемента из пустого стека и переполнения стека. При реализации стека в виде вектора переполнение возникает вследствие ограниченности объема памяти, выделяемой под размещение массива. Однако, даже если стек реализован на основе динамического списка, его размер также не безграничен, он ограничен объемом доступной программе оперативной памяти (обычно он ограничивается небольшим объемом, 256, 512 КБ). Для отработки указанных аварийных ситуаций в программе должны быть предусмотрены дополнительные проверки на пустоту и переполнение стека и выдача соответствующих сообщений.

1.2 Реализации стека

Ниже представлены основные варианты реализации стека на основе массива и списка.

1.2.1 Реализация стека в виде массива

Реализация стека в виде массива. Если стек реализован в виде статического или динамического массива (вектора), то для его хранения обычно отводится непрерывная область памяти ограниченного размера, имеющая нижнюю и верхнюю границу. Перед началом работы указатель стека PS находится ниже левой (нижней) границы массива. При включении первого элемента в стек указатель PS устанавливается на начало массива и по адресу первого элемента размещается записываемое значение. При попытке добавить в стек каждый последующий элемент сначала происходит смещение указателя на длину типа данных (т.е. он перемещается к следующему элементу массива), а затем – размещение значений элементов по этим адресам. Если указатель выходит за верхнюю границу массива, то это признак того, что стек переполнен. При исключении элемента из стека сначала считывают-

ся данные, а затем происходит перемещение указателя PS к предыдущему элементу. Если указатель стека выходит за нижнюю границу массива, то стек пуст.

1.2.2 Реализация стека в виде списка

Реализация стека в виде списка. До начала работы указатель стека показывает на нулевой, физически отсутствующий адрес (т. е. указатель - пустой). 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.

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. Если указатель имеет значение нулевого адреса, то стек пуст.

При физической реализации стека в виде односвязного линейного списка дескриптор будет отличаться отсутствием верхней границы стека. В этом случае объем стека ограничивается только объемом доступной оперативной памяти.

1.3 Условие задачи

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) 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.

1.4 Техническое задание

При реализации стека массивом располагать два стека в одном массиве. Один стек располагается в начале массива и растет к концу, а другой располагается в конце массива и растет к началу. Заполнять и освобождать стеки произвольным образом с экрана. Элементами стека являются вещественные числа. Списком реализовать один стек.

1.5 Вывод

В данном разделе были представлены основные теоритические сведения о стеках и их реализациях. Также было сформулировано техническое задание.

2 Конструкторский раздел

В данном разделе описаны используемые структуры данных, а также представлены схемы алгоритмов и подсчет затрат памяти под хранение стеков.

2.1 Схемы алгоритмов

Ниже, на рисунке 2.1, представлена схема алгоритма вставки элемента в стек, представленный списком.

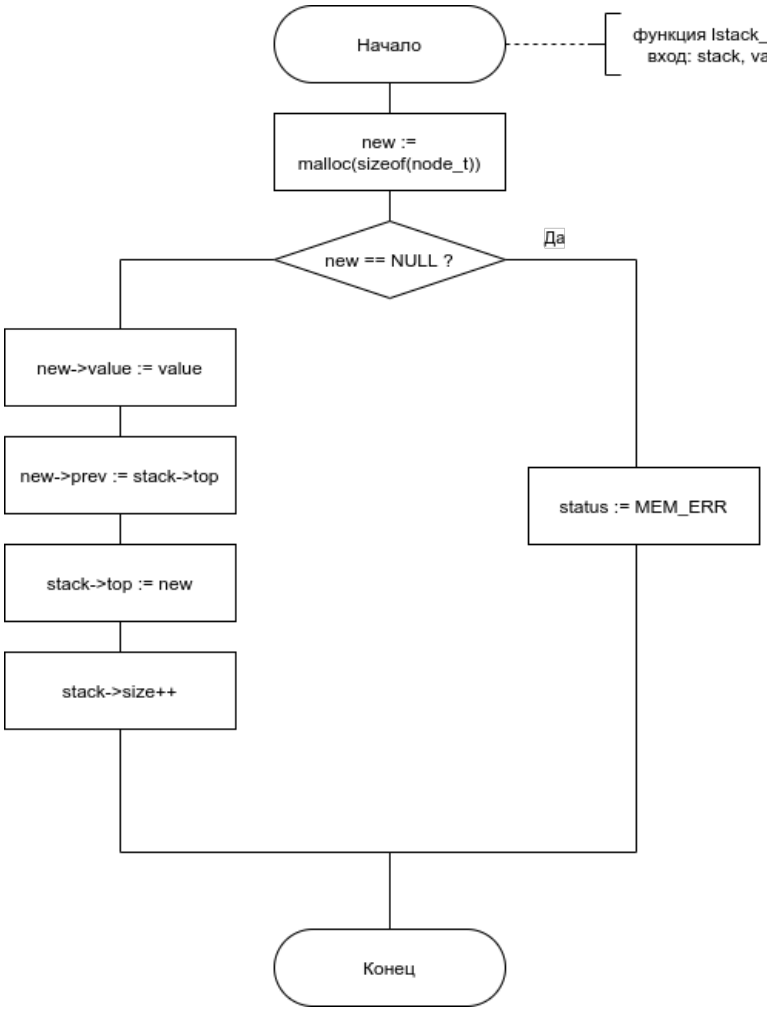

Рисунок 2.1 — схема алгоритма вставки элемента в стек-список.

Ниже, на рисунке 2.2, представлена схема алгоритма удаления элемента из стека, представленного списком.

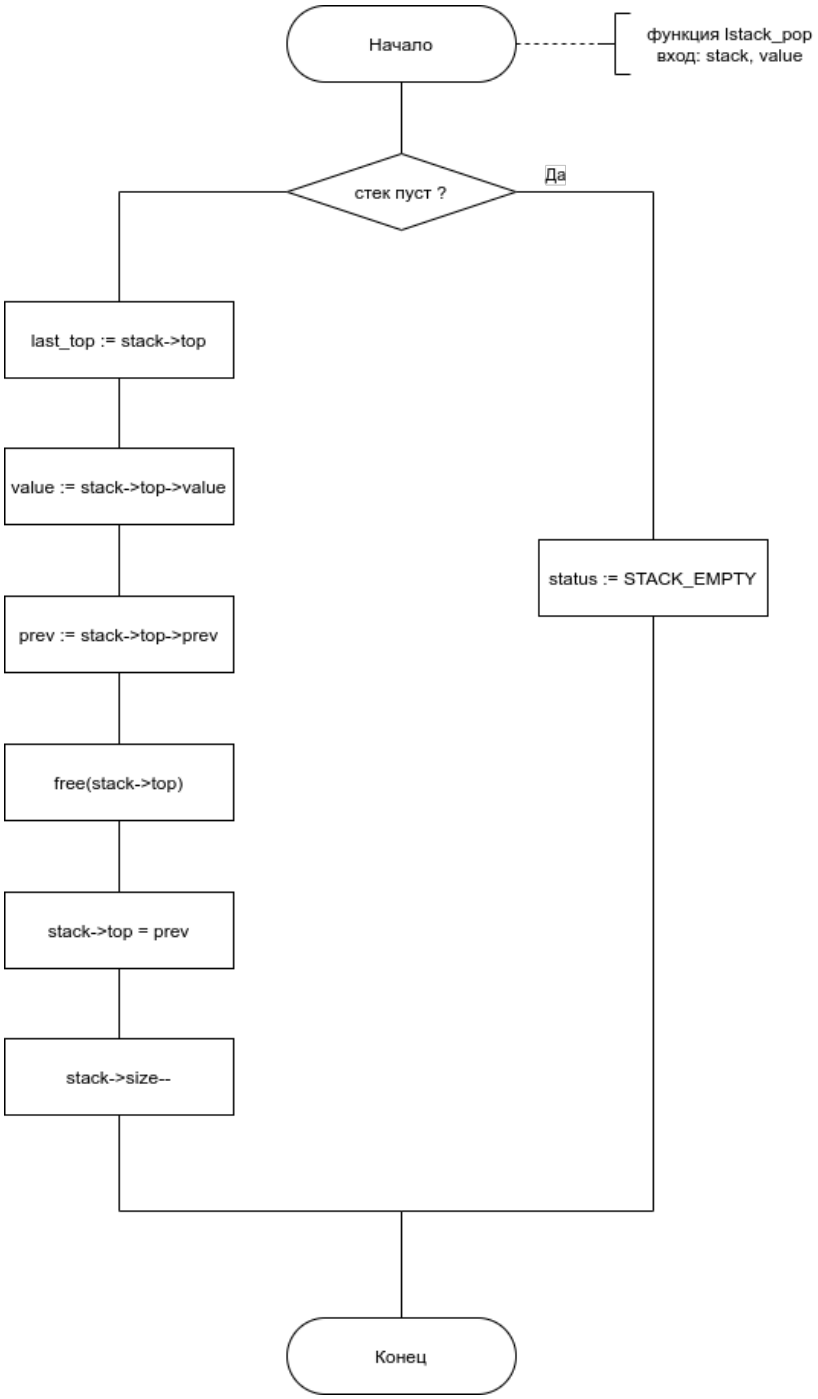

Рисунок 2.2 — схема алгоритма удаления элемента из стека-списка.

Ниже, на рисунке 2.3, представлена схема алгоритма вставки в нижний стек, хранящийся в массиве.

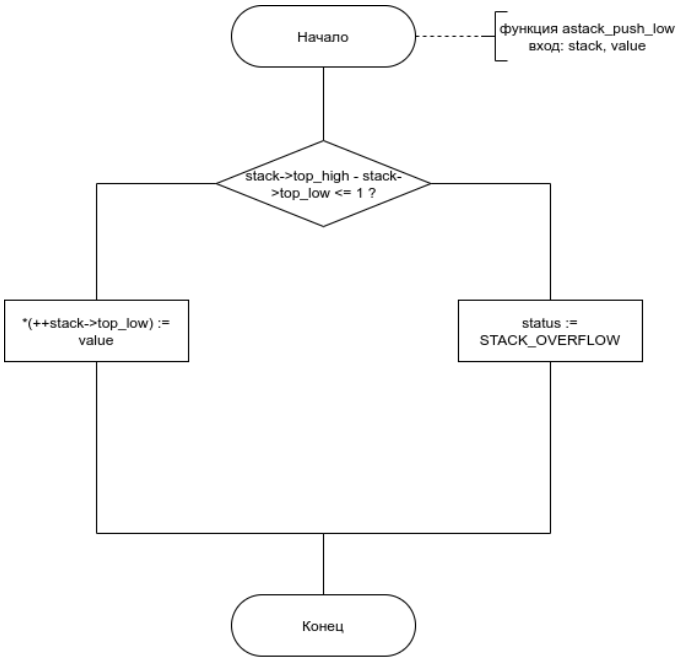

Рисунок 2.3 — схема алгоритма вставки элемента в нижний стек-массив.

Ниже, на рисунке 2.4, представлена схема алгоритма вставки в нижний стек, хранящийся в массиве.

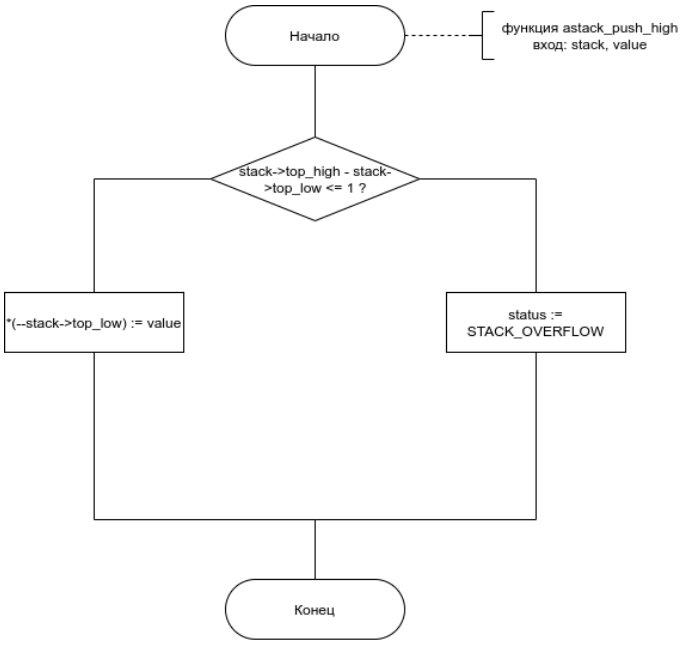

Рисунок 2.4 — схема алгоритма вставки элемента в верхний стек-массив.

Ниже, на рисунке 2.5, представлена схема алгоритма удаления из нижнего стека, хранящегося в массиве.

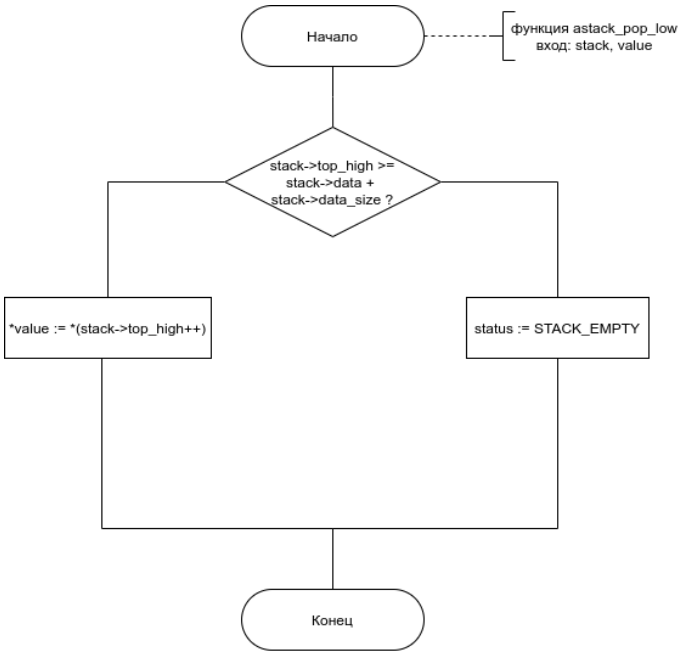

Рисунок 2.5 — схема алгоритма удаления элемента из нижнего стека-массива.

Ниже, на рисунке 2.6, представлена схема алгоритма удаления из верхнего стека, хранящегося в массиве.

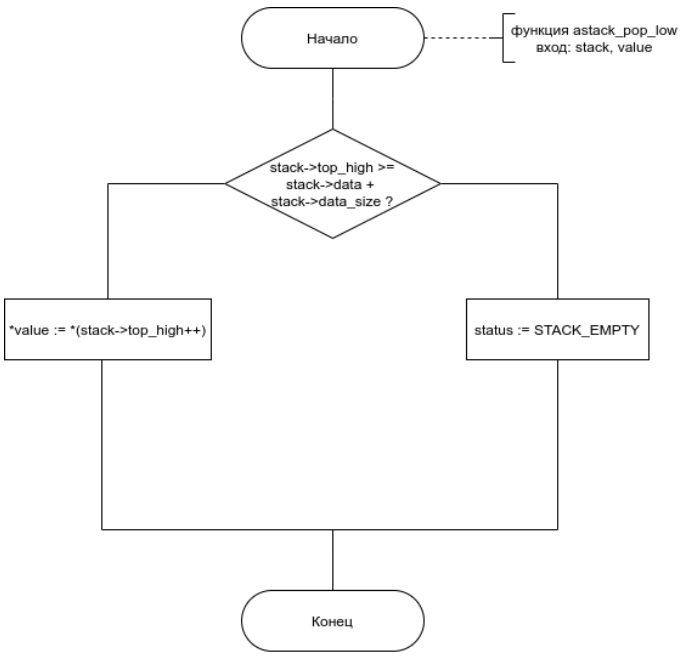

Рисунок 2.6 — схема алгоритма удаления элемента из верхнего стека-массива.

2.2 Описание структур данных

Ниже представлены листинги кодов используемых в ПО структур данных.

Листинг 2.1 — Стек на списке.

```
#define UNBOUND 0        // Если стек, ограничен только объемом оперативной памяти
                         // то capacity = UNBOUND

typedef double data_t;

typedef struct node
{
    struct node *prev;   // Предыдущий (нижний) элемент
    data_t value;        // Значение текущего элемента
} node_t;

typedef struct
{
    node_t *top;         // Верхний элемент стека
    size_t size;         // Количество элементов в стеке
    size_t capacity;     // Вместимость стека
} lstack_t;
```

Листинг 2.2 — Стек на массиве

```
typedef double data_t;

typedef struct
{
    data_t *data;        // Указатель на область памяти
    data_t *top_low;     // Указатель на верхний элемент нижнего стека
    data_t *top_high;    // Указатель на верхний элемент верхнего стека
    size_t data_size;    // Размер массива
} astack_t;
```

Листинг 2.3 — Коды ошибок

```
#define SUCCESS          0   // Успешное выполнение
#define MEM_ERR          1   // Ошибка выделения памяти
#define INP_ERR          2   // Ошибка ввода
#define STACK_OVERFLOW   3   // Переполнение стека
#define STACK_EMPTY      4   // Пустой стек
#define STACK_INVALID    5   // Некорректный стек
```

Листинг 2.4 — Список адресов памяти для стека реализованного списком

```
typedef struct mem_list
{
    const void *addr;        // Адрес памяти
```

```
4       memstate_t state;       // Состояние адреса
5       struct mem_list *next;  // Следующий адрес
6   } mem_list_t;
7
8   // Состояние адресов памяти
9   typedef enum
10  {
11      UNUSED,     // Неиспользован
12      USED,       // Использован
13      REUSED      // Переиспользован
14  } memstate_t;
```

2.3 Вывод

В данном разделе были представлены схемы операций над стеками, а также приведены листинги кодов используемых структур данных.

3 Технологический раздел

В данном разделе будут представлены листинги кодов набора функций для работы со стеком, а также будет приведены требования к ПО и тестовые данные.

3.1 Требования к ПО

Ниже определены предъявляемые к ПО требования:

- На вход программа получает пункт меню (от 0 до 8) и элемент для вставки при необходимости (вещественный);

значения пунктов меню:

0 — выход;

1 — вывод стеков в массиве;

2 — вывод стека на списке;

3 — вставка в нижний стек в массиве;

4 — вставка в верхний стек в массиве;

5 — вставка в стек на спике;

6 — удаление из нижнего стека в массиве;

7 — удаление из верхнего стека в массиве;

8 — удаление из стека на списке.

При выборе пунктов меню 3, 4 или 5: программа получает на вход вещественное число

- На выход программа подает текущее состояние стеков и адреса высвобождаемых/занятых/-переиспользуемых списком областей памяти, а также время вставки элемента и используемую под каждую реализацию стека память.

3.2 Набор функций для работы со стеком

3.2.1 Стек в виде списка

Ниже представлен набор функций для работы со стеком на списке.

Листинг 3.1 — Набор функций для стека-списка.

```
1  // Пустой список
2  lstack_t lstack_empty(size_t capacity);
3
4  // Освобождение памяти
5  void lstack_destroy(lstack_t *stack);
6
7  // Проверка на пустоту
8  bool lstack_is_empty(const lstack_t *stack);
9
10 // Вставка элемента
11 int lstack_push(lstack_t *stack, data_t value);
12
```

```
13 // Удаление элемента
14 int lstack_pop(lstack_t *stack, data_t *value);
15
16 // Вывод стека и адресов
17 void lstack_print(const lstack_t *stack);
18
19 // Память, занимаемая стеком
20 size_t lstack_sizeof(const lstack_t *stack);
```

3.2.2 Стек в виде массива

Ниже представлен набор функций для работы со стеками в массиве

Листинг 3.2 — Набор функций для стека-массива.

```
1  // Непроинициализированный стек
2  astack_t astack_null(void);
3
4  // Пустой стек
5  astack_t astack_empty(size_t size);
6
7  // Освобождение памяти
8  void astack_destroy(astack_t *stack);
9
10 // Проверка на корректность
11 bool astack_is_valid(const astack_t *stack);
12
13 // Вставка в нижний (первый) стек
14 int astack_push_low(astack_t *stack, data_t value);
15
16 // Вставка в верхний (второй) стек
17 int astack_push_high(astack_t *stack, data_t value);
18
19 // Удаление из нижнего (первого) стека
20 int astack_pop_low(astack_t *stack, data_t *value);
21
22 // Удаление из верхнего (второго) стека
23 int astack_pop_high(astack_t *stack, data_t *value);
24
25 // Вывод нижнего (первого) стека
26 void astack_low_print(const astack_t *stack);
27
28 // Вывод верхнего (второго) стека
29 void astack_high_print(const astack_t *stack);
30
31 // Вывод используемой стеком памяти
32 size_t astack_sizeof(const astack_t *stack);
```

3.3 Тестовые данные

Ниже представлены тестовые данные для вставки и удаления элемента из стека.

Таблица 3.1 — Тестовые данные

Описание	Ввод	Стек до операции	Стек после операции
Вставка	3 1.91	$()$	(1.91)
Вставка некорректного элемента	3 1.91.12	$()$	Сообщение об ошибке
Удаление	6	$(1.2, 9.4, 0.0, 1.3)$	$(1.2, 9.4, 0.0)$
Удаление из пустого стека	6	$()$	Сообщение об ошибке
Вставка в переполненный стек	3	$(1, 2, 3, 4, 5)$	Сообщение об ошибке

3.4 Вывод

В данном разделе были представлены листинги кодов функций для работы с реализациями стеков на основе списка и массива, а также были представлены требования к разрабатываемому ПО и тестовые данные. Все тесты пройдены успешно.

4 Экспериментальный раздел

В данном разделе будет представлен сравнительный анализ производительности реализаций стеков на основе списка и на основе массива.

4.1 Постановка эксперимента

В данном эксперименте производится сравнительный анализ производительности реализаций стеков на основе списка и на основе массива. Анализ производится по времени работы и по объему затрачиваемой памяти.

4.1.1 Технические характеристики

Ниже представлены технические характеристики устройства, на котором было произведено тестирование ПО:

- операционная система: Ubuntu 20.04 Linux 64-bit;
- оперативная память: 16GB;
- процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx @ 8x 2,1GHz.

4.1.2 Описание экспериментальных данных

Тестирование будет проведено на основе стеков, максимальный размер которых 25 элементов. Для тестирования временной эффективности будет произведено 10 замеров вставки-удаления элемента в стек, после чего результат будет усреднен.

4.2 Результат эксперимента

4.2.1 Время работы

В результате подсчета эффективности по времени для стеков на основе массива и списка, были получены следующие данные, представленные ниже, на таблице 4.1.

Таблица 4.1 — Временная эффективность реализаций стеков

Реализация	Вставка, тики	Удаление, тики
Список	3318	1762
Массив	483	588

Как видно из таблицы, реализация стека на основе массива примерно в 7 раз быстрее реализации стека на основе списка по операции добавления, и примерно в 3 раза быстрее по операции удаления.

4.2.2 Затраты памяти

Ниже, на рисунке 4.1, представлен график эффективности реализаций стека на основе массива и на основе списка по памяти. Шкала справа от графика расчитывается по следующей формуле:

$$f = \frac{V_A - V_L}{V_A + V_L} \tag{4.1}$$

где V_A объем памяти, занимаемый стеком на основе массива, а V_L на основе списка

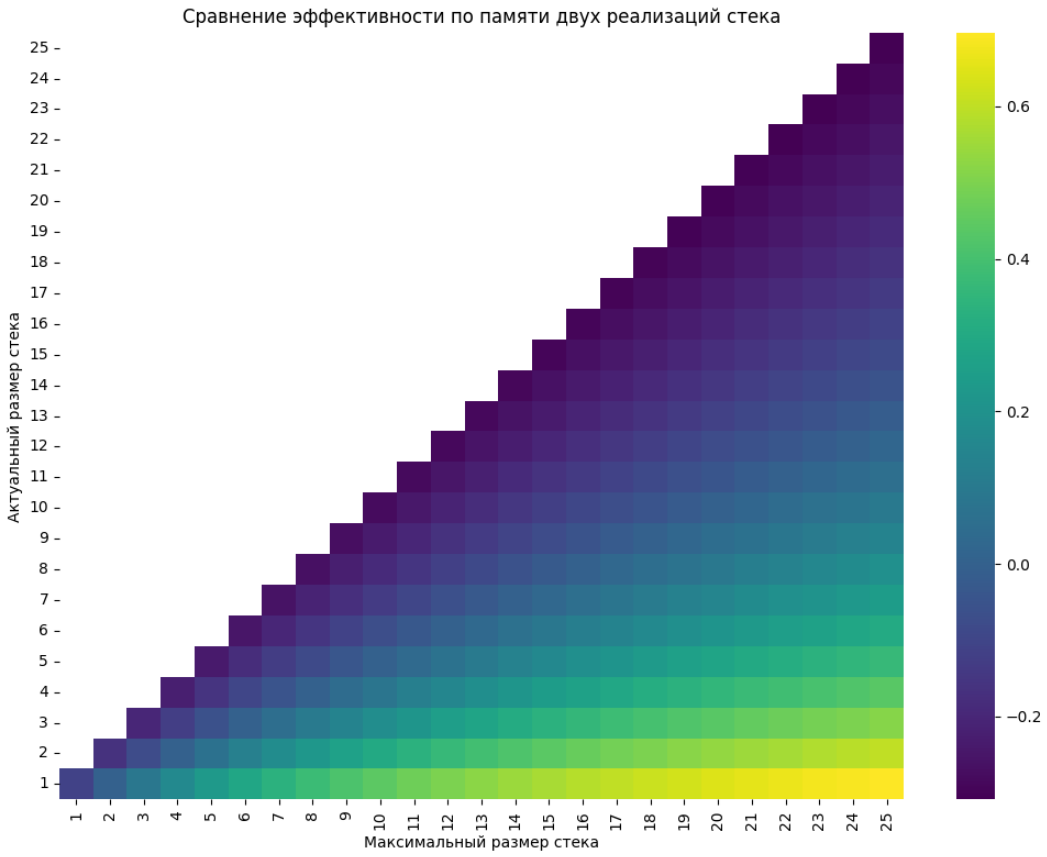

Рисунок 4.1 — Эффективность по памяти

Ниже, на таблице 4.2, представлена таблица сравнения эффективности по памяти стеков на основе массива и на основе списка. По результатам тестирования было выявлено, что при неполной заполненности стека (менее 50%) эффективнее использовать реализацию на основе списка, так как для хранения стека потребуется меньший объем памяти.

Таблица 4.2 — Эффективность по памяти

Макс. размер	процент заполнения, %	массив, байт	список, байт	эффективность, %
10	10	104	40	62
10	50	104	104	0
10	70	104	136	-31
1000	20	8024	3224	60
1000	40	8024	6424	20
1000	50	8024	8024	0
1000	60	8024	9624	-20

4.3 Вывод

В результате эксперимента было получено, что операции со стеком, реализованном на основе массива в 7 раз эффективнее по времени, чем со стеком, реализованном на основе списка. Также было получено, что при заполненности стека менее 50% стек на списке занимает меньше памяти, чем стек в массиве. В результате можно сделать вывод, что реализацию стека на списке выгодно использовать только в том случае, если необходимо получить стек, ограниченный лишь объемом оперативной памяти и если размер стека не превосходит половины от максимального.

5 Контрольные вопросы

1) Что такое стек?

Стек — это абстрактная структура данных, имеющая операции вставки и удаления элементов стека. Работает по принципу LIFO – последний пришёл - первый вышел. (работаем только с вершиной стека)

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

При реализации стека на статическом массиве память выделяется на стеке (машинном) единожды и никак не меняет свой размер во время работы программы.

При реализации стека на динамическом массиве память выделяется в куче единожды в момент инициализации и впоследующем при нехватке свободного места в стеке.

При реализации стека на связном списке память выделяется каждый раз при добавлении нового элемента в стек.

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

При реализации стека на статическом массиве память очищается при выходе из области видимости стека, или проще говоря при завершении работы со стеком.

При реализации стека на динамическом массиве память очищается при необходимости сжать стэк (уменьшить его вместимость до действительного размера), но этого можно не делать при частых и сбалансированных вставках и удалениях.

При реализации стека на связном списке память очищается каждый раз при удалении элемента из стека.

4) Что происходит с элементами стека при его просмотре?

В классической реализации стека просмотр (как элементарное действие над стеком) невозможен. Для просмотра стека необходимо использовать дополнительную структуру данных (не обязательно другой стек), переместив в неё последовательно все элементы из вершины стека. Тогда, складывая элементы обратно в стек, мы будем наблюдать их порядок при добавлении.

5) Каким образом эффективнее реализовывать стек?

Эффективнее реализовывать стек на массиве, когда точно известно максимальное число элементов в нём, а размер стека зачастую будет превышать половину от максимального. Но если не представляется возможным вычислить это число заранее или зачастую в стеке будет храниться не так много элементов (менее 50%), то эффективнее будет использовать стек на связном списке.

Заключение

В результате проделанной работы были исследованы варианты реализации стеков, спроектированы схемы алгоритмов вставки и удаления из стека для различных реализаций, описаны структуры данных. В итоге было разработано ПО, на основе которого проведено сравнение эффективности по времени и памяти реализаций стеков через список и через массив.

В ходе сравнительного анализа упомянутых выше реализаций стеков было выявлено:

1) Стек на массиве работает в 7 раз быстрее стека на списке.
2) Стек на массиве эффективнее по памяти при заполненности более 50%.
3) Стек на списке эффективнее по памяти при заполенности менее 50%.
4) Стек на списке эффективнее, если необходимо не ограничивать размер стека, а ограничиваться лишь объемом оперативной памяти.