|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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

ДИСЦИПЛИНА Типы и структуры данных

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4 «РАБОТА СО СТЕКОМ».**

Студент Иванов Павел Александрович

Группа ИУ7 – 35Б

Вариант №3

Проверил

**Оглавление**

[Цель работы 2](#_Toc86403446)

[Описание технического задания 3](#_Toc86403447)

[Описание структур данных 5](#_Toc86403448)

[Описание алгоритма 7](#_Toc86403449)

[Набор тестов 8](#_Toc86403450)

[Оценка эффективности 9](#_Toc86403451)

[Ответы на контрольные вопросы 10](#_Toc86403452)

# **Цель работы**

Реализовать операции 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.

Описание условия 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.

Все стандартные операции со стеком должны быть оформлены отдельными подпрограммами. В случае реализации стека в виде списка при отображении текущего состояния стека предусмотреть возможность просмотра адресов элементов стека и создания дополнительного собственного списка свободных областей (адресов освобождаемой памяти при удалении элемента, который можно реализовать как списком, так и массивом) с выводом его на экран. Список свободных областей необходим для того, чтобы проследить, каким образом происходит выделение памяти менеджером памяти при запросах на нее и убедиться в возникновении или отсутствии фрагментации памяти.

Элементами стека являются слова. Распечатайте слова в обратном порядке в перевёрнутом виде.

Указания к выполнению работы:

Интерфейс программы должен быть понятен неподготовленному пользователю. При разработке интерфейса программы следует предусмотреть:

* указание формата и диапазона вводимых данных,
* блокирование ввода данных, неверных по типу,
* указание операции, производимой программой:
* добавление элемента в стек,
* удаление элемента из стека,
* вычисление (обработка данных);
* наличие пояснений при выводе результата.

Кроме того, нужно вывести на экран время выполнения программы при реализации стека списком и массивом, а также указать требуемый объем памяти. Необходимо так же выдать на экран список адресов освобождаемых элементов при удалении элементов стека.

При тестировании программы необходимо:

* проверить правильность ввода и вывода данных (в том числе, отследить попытки ввода данных, неверных по типу);
* обеспечить вывод сообщений при отсутствии входных данных («пустой ввод»);
* проверить правильность выполнения операций;
* обеспечить вывод соответствующих сообщений при попытке удаления элемента из пустого стека;
* отследить переполнение стека.

При реализации стека в виде списка необходимо:

* ограничить доступный объем оперативной памяти путем указания:
  + максимального количества элементов в стеке;
  + максимального адреса памяти, превышение которого будет свидетельствовать о переполнении стека;
* следить за освобождением памяти при удалении элемента из стека.

# **Описание технического задания**

Входные данные:

*Число:*

Целое число от 0 до 13 — пункт меню или число добавляемых в стек элементов.

*Строка:*

Слово без пробелов не более 100 символов — элемент стека.

Выходные данные:

Строки (данные, хранящиеся в стеке), адреса, свободные адреса, число тактов выполнения операций, память в байтах, которую занимают стеки, перевёрнутые слова в обратном порядке.

Действие программы:

Выполнение операций со стеком: добавление элемента, извлечение, печать, очистка. Решение задачи – вывод перевёрнутых слов в обратном порядке.

*Функции программы:*

Работа со стеком — массивом:

1 – Ввести элементы

2 – Добавить элемент

3 – Извлечь элемент

4 – Печать стека

5 – Очистка стека

Работа со стеком — списком:

6 – Ввести элементы

7 – Ввести элемент

8 – Извлечь элемент

9 – Печать стека

10 – Очистить стек

11 – Вывести свободные адреса

Выданное задание:

12 – Вывести перевёрнутые слова в обратном порядке

Сравнение эффективности:

13 – Вывести сравнение по времени и памяти

Обращение к программе:

Запускается файл app.exe. Если его нет, из консоли следует вызывать make run.

Аварийные ситуации:

* При очистке стек пуст;

Результат: сообщение «Stack is already empty!»

* При извлечении элемента стек пуст;

Результат: сообщение «Stack is already empty!»

* Переполнение стека при добавлении;

Результат: сообщение «Stack overflow!»

* Введено не валидное число элементов в стеке;

Результат: сообщение «Must be a positive number!»

# **Описание структур данных**

Для хранения стека — массива используется следующая структура:

typedef struct arr\_stack\_t

{

void \*top;

void \*bottom;

char \*\*ps;

} arr\_stack;

Поля структуры:

void \*top – указатель на конец массива (void \* для того, чтобы ограничить произвольный доступ)

void \*bottom – указатель на начало массива (void \* для того, чтобы ограничить произвольный доступ)

char \*\*ps (pointer stack) – указатель на элемент массива, элемент которого – указатель на char (указатель на строку)

Для хранения стека — списка используется следующая структура:

typedef struct list\_stack

{

int index;

char \*data;

struct list\_stack \*prev;

} list\_stack;

Поля структуры:

int index — номер текущей ячейки стека - списка

char \*data — указатель на строку

struct list\_stack \*prev – указатель на предыдущий элемент в стеке

Для хранения списка свободных адресов используется:

typedef struct free\_mem

{

size\_t data;

struct free\_mem \*next;

} free\_mem;

Поля структуры:

size\_t data – адрес

struct free\_mem \*next – следующий элемент списка

*Функции программы:*

// инициализация стека - массива

int arr\_stack\_init(arr\_stack \*stack);

// добавление элемента в стек - массив

int arr\_stack\_push(arr\_stack \*stack, char \*str);

// извлечение элемента из стека - массив

char \*arr\_stack\_pop(arr\_stack \*stack);

// вывод стека - массива

void arr\_stack\_print(arr\_stack stack);

// очитска стека - массива

void arr\_stack\_clear(arr\_stack \*stack);

// инициалиация стека - списка

int list\_stack\_init(list\_stack \*\*stack);

// добавление элемента в стек - список

int list\_stack\_push(list\_stack \*\*stack, char \*str, free\_mem \*mem);

// извлечение элемента из стека - списка

char \*list\_stack\_pop(list\_stack \*\*stack, free\_mem \*mem);

// освобождение памяти стека - списка

void list\_stack\_free(list\_stack \*\*stack, free\_mem \*mem);

// печать стека - списка

void list\_stack\_print(list\_stack \*stack);

// инициализия списка свободных областей

void free\_mem\_init(free\_mem \*mem);

// добавление адреса в список свободных адресов

int free\_mem\_add(free\_mem \*mem, size\_t add);

// удаление адреса из списка свободных адресов при его наличии

void free\_mem\_del(free\_mem \*mem, size\_t del);

// очистка стека - списка

void list\_stack\_clear(list\_stack \*\*stack, free\_mem \*mem);

// вывод списка свободных адресов

void free\_mem\_print(free\_mem \*mem);

# **Описание алгоритма**

1. Ввод номера команды;
2. При работе с массивом:
   1. Добавление: смещается указатель, осуществляется проверка выхода за границы, записывается элемент.
   2. Извлечение: указатель смещается вниз, осуществляется проверка выхода за границы, возвращается удалённый элемент.
3. При работе со списком:
   1. Добавление: выделяется память, элемент помещается в начало списка
   2. Извлечение: возвращаются данные, список теперь начинается со второго элемента
4. Вывод перевёрнутых слов в обратном порядке: слова кладутся на стек и извлекаются. Буквы текущего слова кладутся на стек и при извлечении печатаются.

# **Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовый случай | Пользовательский ввод | Результат |
| 1 | Ввод некорректного числа элементов | a | Must be a positive number! |
| 2 | Ввод некорректного числа элементов | -10 | Must be a positive number! |
| 3 | Ввод некорректного числа элементов | 0 | Must be a positive number! |
| 4 | Печать пустого стека | 4 / 9 | The stack is empty! |
| 5 | Извлечение элемента из пустого стека | 5 / 8 | The stack is empty! |
| 6 | Добавление элемента в полностью заполненный стек | 2 / 7 | Stack overflow! |
| 7 | Добавление элементов | 1 / 6 — 5 a b c d e | В стек добавлены 5 элементов |
| 8 | Вывод стека | 4 / 9 | Выведены 5 элементов a b c d e |
| 9 | Добавление элемента | 2 / 7 — word | Теперь в стеке 6 элементов |
| 10 | Очистка стека | 5 / 10 | Стек пустой |
| 11 | Вывод списка свободных адресов | 11 | Все освобождённые адреса выведены на экран |
| 12 | Вывод перевёрнутых слов в обратном порядке | 5  abc def ghj a l | l a jhg fed cba |
| 13 | Вывод перевёрнутых слов в обратном порядке | 1 a | a |
| 14 | Вывод измерения времени | 13 | На экране выведена таблица времени и памяти |

# **Оценка эффективности**

Оценка эффективно­­сти проводится по операциям добавления элемента и извлечения элемента. Измеряется число тактов, усреднённое за 1000 прогонов:

Операция добавления:

|  |  |  |
| --- | --- | --- |
| Число элементов | Стек — массив | Стек — список |
| 10 | 40 | 488 |
| 100 | 34 | 1259 |
| 1000 | 31 | 6774 |

Операция извлечения:

|  |  |  |
| --- | --- | --- |
| Число элементов | Стек — массив | Стек — список |
| 10 | 30 | 320 |
| 100 | 31 | 2964 |
| 1000 | 35 | 23613 |

Память:

|  |  |  |
| --- | --- | --- |
| Число элементов | Стек — массив, байт | Стек — список, байт |
| 10 | 24016 | 240 |
| 100 | 24016 | 2400 |
| 1000 | 24016 | 24000 |

# **Ответы на контрольные вопросы**

1. *Что такое стек?*

Стек — последовательный список с переменной длинной, действия с которым (включение и исключение элементов) происходят только с одной его стороны — с его вершины. Эта структура данных используется по принципу Last In — First Out (LIFO).

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

При реализации с помощью массива память под стек может выделяться как динамически, так и статически (в любом случае выделится непрерывный участок памяти ограниченного размера). Так, выделяется кусок памяти размера sizeof(тип элемента) \* (количество элементов в стеке). В первом случае память выделяется в куче, во втором — на стеке.

При реализации связным списком память выделяется каждый раз, когда добавляется новый элемент. Таким образом, число элементов в стеке ограничено только объёмом оперативной памяти.

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

В случае с массивом, если память статическая, то происходит возврат текущего значения и переход указателя вниз на один элемент (память не освобождается). Если же память динамическая, то способ освобождения зависит от реализации: можно заново выделить память нового размера, освободив старую, причем заранее скопировав данные оттуда в новую. А можно работать так же, как со статическим, и освободить все ресурсы в конце программы.

В случае со списком, при удалении элемента нужно вернуть значение текущей ячейки, записать нулевой указатель в предыдущую ячейку и освободить память из-под текущей.

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

Все элементы стираются, так как чтобы увидеть последний элемент (предыдущие просто недоступны), надо его достать.

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

По результатам оценки эффективности видно, что стек в виде массива использовать гораздо эффективнее по времени. По памяти же эффективнее использовать список, если мы не знаем максимальное количество элементов в стеке (либо это количество сильно варьируется): тогда придется выделять большой кусок памяти, который может и не понадобиться, но он будет занимать излишнюю память.

Вывод

В ходе данной работы я познакомился со структурой данных «стек», разобрался с операциями, которые можно реализовывать на этой структуре и сравнил реализацию стека списком и массивом.

Получилось, что если важно время работы, то выгоднее использовать стек на массиве. Стек на списке же удобнее использовать когда мы не знаем максимальное количество элементов в стеке (либо это количество сильно варьируется): тогда придется выделять большой кусок памяти, который может и не понадобиться, но он будет занимать излишнюю память.