|  |  |
| --- | --- |
|  | **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 предмета: Типы и структуры данных

Студент: Гриценко Алексей Михайлович

Группа: ИУ7-31Б

*2020г.*

1. **Описание условия задачи**
2. Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека.

Реализовать стек:

массивом;

списком.

1. Все стандартные операции со стеком должны быть оформлены подпрограммами. При реализации стека списком в вывод текущего состояния стека, создать свой список или массив свободных областей (адресов освобождаемых элементов) с выводом его на экран.
2. Распечатать убывающие серии последовательности целых чисел в обратном порядке.
3. **Техническое задание**
4. ***Описание исходных данных***

Целое число, представляющее собой номер команды: целое число в диапазоне от 0 до 11.

Командно-зависимые данные: *целочисленные значения (Значение элементов стека)*

1. ***Описание результата программы***

Результат выполнения индивидуального задания – убывающие подпоследовательности в исходной последовательности.

*Количественная характеристика* сравнения *вариантов обработки стека.*

1. ***Задача, реализуемая программой***
2. Ввести элементы стека.
3. Добавить элемент в стек.
4. Удалить элемент из стека.
5. Вывести убывающие подпоследовательности в обратном порядке и вывести количественную характеристику обработки.
6. Вывести текущее состояние стека.

п. 1-5 выполняют действия над стеком, реализованным на основе массива.

1. Ввести элементы стека.
2. Добавить элемент в стек.
3. Удалить элемент из стека.
4. Вывести массив освободившихся адресов.
5. Вывести убывающие подпоследовательности в обратном порядке и вывести количественную характеристику обработки.
6. Вывести текущее состояние стека.

п. 6-11 выполняют действия над стеком, реализованным на основе связного списка.

1. ***Обращения к программе***

Способ обращения к программе происходит через консоль.

1. ***Описание возможных аварийных ситуаций и ошибок пользователя***
2. Некорректный ввод номера команды.  
   На входе: число, большее чем 11 или меньшее, чем 0.   
   На выходе: сообщение «Введена недопустимая команда! Повторите попытку.»
3. Некорректный ввод количества элементов стека.  
   На входе: отрицательное целое число, число, превышающее максимально допустимое число для количества элементов стека или буква.  
   На выходе: сообщение «Введено недопустимое значение! Повторите поппытку.»
4. Некорректный ввод элемента стека.  
   На входе: буква или любой другой нечисловой символ.  
   На выходе: сообщение «Введено недопустимое значение! Повторите поппытку.»
5. Попытка создать новый стек, при имеющемся в программе.  
   На входе: попытка создания нового стека.  
   На выходе: сообщение «Стек уже существует. Выход из программы...»
6. Удаление элементов из уже пустого стека, сообщение: Ошибка! В стеке не осталось элементов
7. В стеке слишком много элементов, сообщение : Ошибка! Переполнение стека!
8. **Описание внутренних структур данных**

За хранение стека в виде массива отвечает структура данных Stack\_t:

typedef struct Stack\_tag {

T data[STACK\_MAX\_SIZE];

size\_t size;

} Stack\_t;

Поля структуры:

* T data[STACK\_MAX\_SIZE]  **–**  массив, хранящий сами элементы стека;
* *size\_t size* ***–***количество элементов стека.

За хранение стека в виде односвязного списка, отвечает структура данных Node\_t:

typedef struct Node\_tag {

T value;

struct Node\_tag \*next;

} Node\_t;

Поля структуры:

* T value **–** значение элемента стека;
* struct Node\_tag \*next ***–***Указатель на следующий элемент списка.

За хранение номеров ячеек памяти, высвобожденных при работе со стеком, отвечает структура данных arr\_svob:

typedef struct arr\_svob {

size\_t \*data;

size\_t size\_svob;

} arr\_svob;

Поля структуры:

* \*data **–** массив, хранящий номера ячеек памяти/указатель на массив адресов;
* size\_t size\_svob ***–***размер данного массива.

1. **Описание алгоритма**
2. Пользователь вводит номер команды из меню.
3. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия с двумя реализациями стека – на основе массива или на основе линейного односвязного списка.
4. При вводе стека, стек сразу хранится выбранным способом представления (на основе массива или на основе линейного односвязного списка).
5. При выборе команды добавить/удалить элемент из стека, элемент добавляется/удаляется из стека на основе выбранной реализации.
6. При выборе команды вывода массива свободных областей, выводится массив свободных областей в том случае, если какие-либо элементы были удалены из стека.
7. При выборе команды нахождения убывающих подпоследовательностей, решение реализуется на выбранном представлении и выводится количественная характеристика выполнения задачи. В данной функции идет цикл, пока в списке находятся элементы. Внутри данного цикла, с помощью другого, который смотрит новый элемент и предыдущий, и в зависимости от результата, либо заносит в массив подпоследовательностей, либо идет итерация следующего цикла. В конце печатается полученный массив, либо сообщение об отсутствии подпоследовательностей

**Функции, использующиеся в программе**

int push(Stack\_t \*stack, const T value) - функция добавления элемента на вершину стека на снове массива

T pop(Stack\_t \*stack) – функция удаления элемента с вершины стека на основе массива

T peek(const Stack\_t \*stack) – функция возвращания элемента с вршины

void printStack(const Stack\_t \*stack, void (\*printStackValue)(const T)) – функция печати элементов стека на основе массива

int push\_1(Node\_t \*\*head, T value) – функция добавления элемента на вершину стека на основе списка

T pop2(Node\_t \*\*head, arr\_svob \*mas\_free) – функция удаления элемента с вершины стека на основе списка

void printStack\_list(const Node\_t\* head) – функция печати стека на основе списка

size\_t getSize\_list(const Node\_t \*head) – функция получения размера стека на основе списка

int decsubseq\_arr(Stack\_t \*stack\_copy) – функция нахождения убывающей последовательности и печати ее в обратном порядке для массива

int decsubseq\_list(Node\_t \*\*root) – функция нахождения убывающей последовательности и печати ее в обратном порядке для списка

void list\_free\_all(Node\_t \*head) – функция чистки памяти

1. **Тестирование**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Некорректный ввод комманды | 100 | Введена недопустимая команда! Повторите попытку. |
| 2 | Некорректный ввод количества элементов стека | -1 | Введено недопустимое значение! Повторите попытку |
| 3 | Некорректный ввод количества элементов стека | A | Введено недопустимое значение! Повторите попытку |
| 4 | Некорректный ввод элемента стека | B | Введено недопустимое значение! Повторите попытку |
| 5 | Некорректный ввод элемента стека | 2.5 | Введено недопустимое значение! Повторите попытку |
| 6 | Попытка создания стека при уже существующем | Ввод нового стека | Стек уже существует! Выход из программы... |
| 7 | Извлечение элемента из пустого стека | Пустой стек | Стек пуст! |
| 8 | Корректный ввод элементов стека | 5  1 2 3 4 5 | Стек успешно заполнен. |
| 9 | Добавление элемента в стек при неполном стеке | 4 | Значение успешно помещено в стек. |
| 10 | Удаление элемента при непустом стеке | Удаление | Значение успешно извлечено: **значение пика стека** |
| 11 | Вывод массива освободившихся адресов | Вывод массива | Освободившиеся адреса: **массив свободных адресов** |
| 12 | Отсутствие решения | Нахождение подпоследовательности | Подпоследовательности не найдены. |
| 13 | Наличие решения | 3 2 1 1 1 | 1 2 3 Время: **время** |

1. **Оценка эффективности**

Измерения эффективности реализаций стека будут производиться в единицах измерения – тактах процессора. Для измерения была специально написана ассемблерная функция, поэтому погрешность измерений минимальна. При записи результатов использовалось среднее количество тактов, полученное по результатам 10 измерений.

**Время добавления элемента (в тактах процессора):**

|  |  |  |
| --- | --- | --- |
| Количество элементов | Массив | Список |
| 10 | 556 | 5154 |
| 100 | 1872 | 24626 |
| 1000 | 14728 | 293568 |

**Время удаления элемента (в тактах процессора):**

|  |  |  |
| --- | --- | --- |
| Количество элементов | Массив | Список |
| 10 | 400 | 5636 |
| 100 | 1306 | 20480 |
| 1000 | 12752 | 298348 |

**Объём занимаемой памяти (в байтах):**

|  |  |  |
| --- | --- | --- |
| Количество элементов | Массив | Список |
| 10 | 56 | 160 |
| 100 | 408 | 1600 |
| 1000 | 4016 | 16000 |

Проведем исследование: возьмем максимальный размер стека 200, и проверим, при каком проценте заполнения стек на массиве начинает выигрывать по памяти стек на списке

|  |  |  |
| --- | --- | --- |
| Количество элементов | Массив | Список |
| 20(процент заполнения 10) | 816 | 160 |
| 30(процент заполнения 15) | 816 | 480 |
| 40(процент заполнения 20) | 816 | 640 |
| 50(процент заполнения 25) | 816 | 800 |
| 55(процент заполнения 30) | 816 | 800 |

Получаем, что если процент заполнения меньше 30, выгоднее использовать список. Особенно его выгодно использовать при проценте заполнения меньше 15 процентов, ведь выгода может быть больше чем в два раза

1. **Выводы по проделанной работе**

Стек, реализованный связанным списком, внушительно (приблизительно в 4 раза по сравнению со временем, достигнутым реализацией на основе массива) проигрывает как по времени, так и по памяти в данной реализации, но стоит заметить что мы использовали статический массив, поэтому на должно быть известно максимальное количество элементов. Таким образом, можно сделать вывод, что если нужно реализовать такую структуру данных как стек, то лучше использовать массив, а не связанный список, но именно в том случае, когда мы уверены, что процент заполнения будет высоким. Если же процент заполнения меньше 15, использовать список по памяти гораздо выгоднее.

1. **Ответы на вопросы**

**1. Что такое стек?**

Стек – структура данных, в которой можно обрабатывать только последний добавленный элемент (верхний элемент). На стек действует правило LIFO — последним пришел, первым вышел.

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

При хранении стека с помощью списка, то память всегда выделяется в куче. При хранении с помощью массива, память выделяется либо в куче, либо на стеке (в зависимости от того, динамический массив или статический). Для каждого элемента стека, реализованного списком, выделяется на 4 или 8 байт (на большинстве современных ПК) больше, чем для элемента массива. Эти дополнительные байты занимает указатель на следующий элемент списка. Размер указателя (4 или 8 байт) зависит от архитектуры.

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

При хранении стека связанным списком, верхний элемент удаляется освобождением памяти для него и смещением указателя, указывающего на начало стека. При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека.

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

Элементы стека уничтожаются, так как каждый раз достается верхний элемент стека.

**5. Каким образом эффективнее реализовывать стек? От чего это зависит?**Если мы знаем, что процент заполнения нашего стека будет меньше 30, и нам важна память, нужная для программы, то лучше реализовать стек на основе списка, потому что он точно будет занимать меньше памяти. В иных случаях выгоднее использовать массив, потому что время для работы с массивом меньше чем время работы со списком