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»

Лабораторная Работа №4 «Работа со стеком»

Вариант №4

Студент **Поляков Андрей Игоревич**

Группа **ИУ7-32Б**

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

Студент **Поляков А.И.**

Оценка _____

Описание условия задачи

Реализовать операции работы со стеком, который представлен в виде статического массива 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.

Техническое задание

Исходные данные:

Пункты меню, выраженные целыми числами 0-10, -1, -2, -5, -6. Внутри некоторых пунктов вводятся числа.

Для стеков реализованных с помощью массива:

1. Добавить элемент в стек 1

-1. Взять элемент из стека 1

2. Добавить элемент в стек 2

-2. Взять элемент из стека 2

3. Отсортировать стеки и вывести результат

4. Вывести оба стека

Для стеков реализованных с помощью списка:

5. Добавить элемент в стек 1

-5. Взять элемент из стека 1

6. Добавить элемент в стек 2

-6. Взять элемент из стека 2

7. Отсортировать стеки и вывести результат

8. Вывести оба стека

Общие:

9. Сравнить производительность

10. Информация

0. Выход

Результат:

Стек, массив освобожденных адресов, элемент стека, отсортированный стек из двух введенных, результаты замеров времени работы алгоритмов и оценки их эффективности.

Описание задачи:

Добавление элементов в стеки, реализованные с помощью статических массивов и списков, взятие элементов из них, их вывод (в случае списка - с адресами и массивом освобожденных адресов), сортировка двух стеков в третий, оценка эффективности алгоритмов.

Способ обращения к программе:

Запуск с помощью ./app.exe

Аварийные ситуации и ошибки:

1. Некорректный формат добавляемого элемента
2. Переполнение статического массива
3. Взятие элемента из пустого стека
4. Суммарный размер стеков для сортировки превышает допустимый размер стека
5. Ошибка выделения памяти
6. Не оба стека для сортировки введены
7. Количество элементов введенное для оценки эффективности превышает допустимый размер стека, реализованного с помощью массива

Описание внутренних структур данных

Реализация с помощью статического массива:

```
#define SIZE 10000

struct arr_stack_type
{
    int data[SIZE];
    size_t top;
};
typedef struct arr_stack_type* arr_stack_t;
```

arr_stack_type :

1. data – массив со всеми элементами
2. top – индекс верхнего элемента стека

typedef – в заголовочном файле для реализации стека как абстрактного типа данных

Реализация с помощью односвязного списка:

```
struct node
{
    int data;
    struct node* next;
};
typedef struct node* list_stack_t;
```

node : узел списка

1. data – значение элемента
2. next – указатель на следующий элемент

typedef – в заголовочном файле для реализации стека как абстрактного типа данных

Массив освобожденных адресов:

```
struct freed_stack_type
{
    list_stack_t* arr;
    size_t size;
    size_t capacity;
};
typedef struct freed_stack_type* freed_stack_t;
```

Тоже реализован как абстрактный тип данных

1. arr – динамический массив адрсов
2. size – размер массива
3. capacity – размер текущей выделенной памяти

typedef – в заголовочном файле для реализации как абстрактного типа данных

Описание алгоритмов

1. Отобразить пользователю список команд и дождаться ввода номера нужной команды.
2. Команды добавления элемента в стек, реализованный массивом, считывают элемент и добавляют его в стек: индекс верхнего элемента увеличивается на 1 и на эту позицию записывается элемент.
3. Команды взятия элемента из стека, реализованного массивом, берут элемент из стека и выводят его: считывается верхний элемент и индекс верхнего элемента уменьшается на 1.
4. Команды добавления элемента в стек, реализованный списком, считывают элемент и добавляют его в стек: создается новый узел и в его поле next записывается текущий верхний элемент, затем верхним элементом становится созданный узел.
5. Команды взятия элемента из стека, реализованного списком, берут элемент из стека и выводят его: считывается значение верхнего элемента, верхним становится ыследующий, освобождается память.
6. Команды вывода стека выводят стек.
7. Команды сортировки сортируют два стека в третий, при этом все данные переносятся в третий стек: сначала сортируются два введенных стека, затем они совмещаются в третий.
8. При выборе замеров, считать размер проверяемых стеков, случайно сгенерировать их содержимое, произвести сортировку двух видов стеков, измерив время. Оценить относительную эффективность программы (в процентах) по времени и по используемому объему памяти в зависимости от используемого алгоритма и от объема информации.
9. При выборе вывода информации о программе, вывести всю необходимую информацию.
10. При выборе выхода, закрыть файл и завершить программу.

Тестовые данные

Позитивные тесты

Тест	Входные данные	Результат
Добавить элемент в стек, основанный на массиве	Стек, элемент	Стек, верхним элементом которого является добавленный элемент
Добавить элемент в стек, основанный на списке	Стек, элемент	Стек, верхним элементом которого является добавленный элемент
Взять элемент из стека, основанного на массиве	Стек	Элемент, стек, верхним элементом которого является следующий элемент
Взять элемент из стека, основанного на списке	Стек, массив освобожденных элементов	Элемент, стек, верхним элементом которого является следующий элемент, массив освобожденных элементов с адресом освобожденного элемента
Отсортировать два стека, основанных на массивах, в третий	Два стека	Два пустых стека и отсортированный стек, содержащий все данные
Отсортировать два стека, основанных на списках, в третий	Два стека	Два пустых стека и отсортированный стек, содержащий все данные
Вывести два стека, основанных на массивах	Два стека	Два пустых стека и отсортированный стек, содержащий все данные
Вывести два стека, основанных на списках вместе с адресами и массивом освобожденных адресов	Два стека, массив освобожденных элементов	Элементы двух стеков, их адреса и освобожденные адреса
Сравнить производительность	Размер данных для теста	Результаты теста

Негативные тесты

Неверная команда в меню	23	Неверный выбор. Попробуйте снова.
Нечисловой элемент	a	Ошибка ввода
Нецелый элемент	2.35	Ошибка ввода
Взятие из пустого стека	Стек	Стек пуст
Добавление 10001-го элемента в стек, основанный на массиве	Стек, элемент	Переполнение
Попытка сортировки без введенных обоих стеков	Пустой стек	Не все стеки введены
Два стека, основанных на массиве, суммарный размер которых больше 10000	Два стека	Суммарный размер стеков превышает допустимый размер стека

Замеры

Программа замеряет время сортировки двух видов стеков. Для каждого из тестов программа выполняет многочисленные замеры, пока их RSE не становится <5. Затем все тестовые случаи сравниваются и оценивается их эффективность.

Количество элементов для сортировки: 100

Реализация с помощью массива:

```
|  Время, нс   | Кол-во итераций | RSE
|    58992.50 |              10 | 1.32
```

Занимаемая память - 40008 байт

Реализация с помощью списка:

```
|  Время, нс   | Кол-во итераций | RSE
|    78551.75 |              40 | 3.93
```

Занимаемая память - 1200 байт

Разность производительности = 19559.25 нс
Реализация с помощью списка дольше на 33.155486 %

Разность занимаемой памяти = 38808 байт
Реализация с помощью массива больше на 3234 %

Количество элементов для сортировки: 500

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
555364.77	30	3.97

Занимаемая память - 40008 байт

Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
1675739.95	20	3.90

Занимаемая память - 6000 байт

Разность производительности = 1120375.18 нс
Реализация с помощью списка дольше на 201.736813 %

Разность занимаемой памяти = 34008 байт
Реализация с помощью массива больше на 566 %

Количество элементов для сортировки: 1000

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
2101599.70	20	4.88

Занимаемая память - 40008 байт

Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
6161988.20	30	3.47

Занимаемая память - 12000 байт

Разность производительности = 4060388.50 нс
Реализация с помощью списка дольше на 193.204657 %

Разность занимаемой памяти = 28008 байт
Реализация с помощью массива больше на 233 %

Количество элементов для сортировки: 3000

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
16995375.30	10	0.12

Занимаемая память - 40008 байт

Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
52948805.40	10	0.58

Занимаемая память - 36000 байт

Разность производительности = 35953430.10 нс
Реализация с помощью списка дольше на 211.548315 %

Разность занимаемой памяти = 4008 байт
Реализация с помощью массива больше на 11 %

Количество элементов для сортировки: 5000

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
46531744.70	10	0.15

Занимаемая память - 40008 байт
Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
145892689.80	10	0.31

Занимаемая память - 60000 байт

Разность производительности = 99360945.10 нс
Реализация с помощью списка дольше на 213.533676 %

Разность занимаемой памяти = 19992 байт
Реализация с помощью списка больше на 49 %

Количество элементов для сортировки: 10000

Реализация с помощью массива:

Время, нс	Кол-во итераций	RSE
186277845.80	10	0.20

Занимаемая память - 40008 байт
Реализация с помощью списка:

Время, нс	Кол-во итераций	RSE
588468860.20	10	0.27

Занимаемая память - 120000 байт

Разность производительности = 402191014.40 нс
Реализация с помощью списка дольше на 215.909204 %

Разность занимаемой памяти = 79992 байт
Реализация с помощью списка больше на 199 %

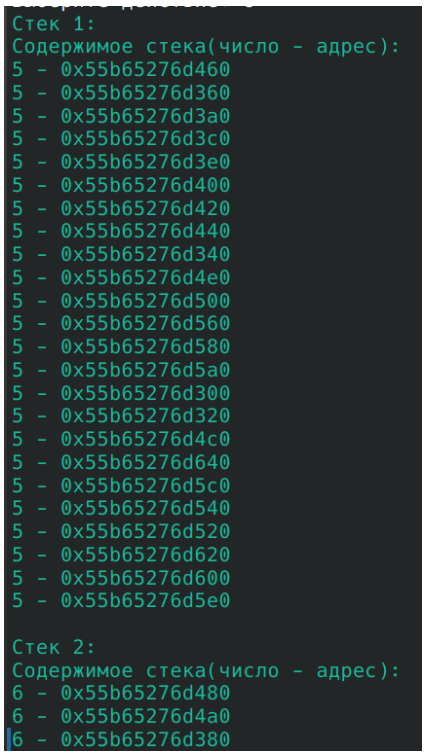
Во всех тестовых случаях реализация с помощью массива оказалась быстрее (в среднем на 65%)

Реализация с помощью списка выигрывала по памяти до 3000 элементов в следствие статичности массива (память выделялась сразу под 10000 элементов)

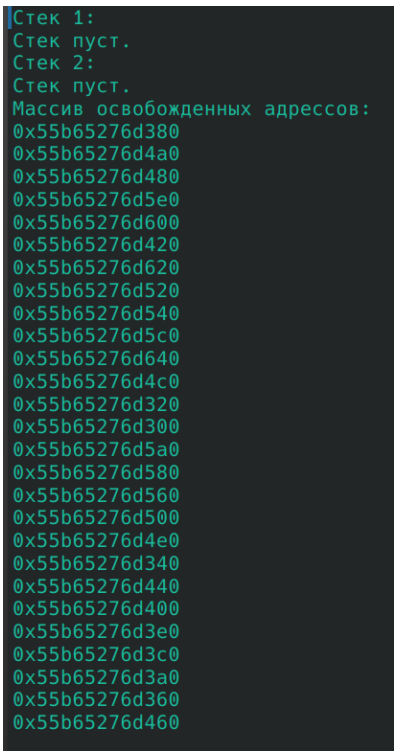
Анализ механизмов выделения и освобождения памяти при работе со стеком

Программа создает массив освобожденных адресов для стека реализованного с помощью списка, при этом, если память выделяется на ранее освобожденном месте в памяти, то этот адрес удаляется из массива. С помощью такой реализации можно отследить наличие или отсутствие процесса, который называется фрагментацией:

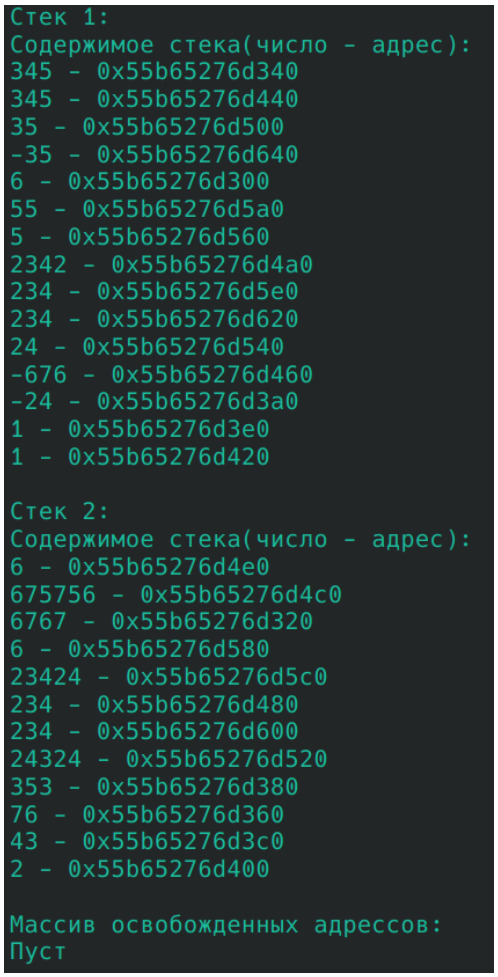
Поочередно буду записывать и считывать числа в стеки и прослежу за адресами:



->>



->>



Все выделенные, затем освобожденные адреса были заново использованы, следовательно, при использовании этих алгоритмов фрагментация памяти не происходит.

Ответы на вопросы

- Стек - это абстрактная структура данных, которая представляет собой упорядоченную коллекцию элементов, где доступ к элементам осуществляется только с одного конца, называемого вершиной. Последний вошедший элемент является первым, который выходит (принцип "Last In, First Out" - LIFO).
- При реализации стека через статический массив память выделяется сразу под всю структуру – под массив целых чисел и под индекс первого элемента. В случае реализации с помощью односвязного списка память выделяется по мере добавления новых элементов и для каждого элемента выделяется память для хранения значения и ссылки на следующий элемент.
- При реализации через статический массив память освобождается для всей структуры сразу. При реализации через односвязный список необходимо пройтись по всем узлам, последовательно освобождая память каждого.
- Для просмотра стека реализована команда pop, которая считывает верхний элемент из стека, вследствие чего стек теряет этот элемент.
- Сравнение реализации стека через массив и через список показало, что эффективнее реализация через массив, так как нет необходимости выделять память для каждого нового элемента стека.

Вывод

В лабораторной работе сравнивались производительности стеков на основе массива и односвязного списка. Стек на основе массива оказался более эффективным в большинстве случаев. Время выполнения операций в нем остается сравнительно низким даже при больших размерах стека. Связанный список становится менее эффективным с увеличением размера стека. Однако стек на основе статического массива ограничен максимальным количеством элементов. Из всего этого можно сделать вывод, что в большинстве ситуаций более предпочтительной будет реализация через массив, за исключением случаев, когда необходимо работать с редко изменяющимися небольшими количествами данных. Также изучив механизмы выделения и освобождения памяти при работе со стеком представленным в виде списка, я отследил, что фрагментация памяти не происходит, что является плюсом, так как память используется эффективнее.