МГТУ им. Н. Э. Баумана

**Дисциплина типы и структуры данных**

**Лабораторный практикум №4**

**по теме: «Обработка разреженных матриц»**

**Вариант №4**

Работу выполнил:

студент группы ИУ7-35

Платонова Ольга

Работу проверил:

Москва, 2019 г.

***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

***Условие задачи***

Создать программу работы со стеком, выполняющую операции добавления, 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 с выводом его на экран.

***Техническое задание***

1. Проверить корректность введенной строки.

2. Организовать стек в виде массива/списка.

3. Организовать возможность работы со стеком

а) добавлене

б) удаление

в) вывод на экран

4. Произвести анализ по времени, памяти.

*Входные данные:*

Пользователь может выбрать в меню

Check expression.......1 (Проверить корректность скобок введенного выражения)

Work with stack........2 (Работать со стеком)

Time...................3 (Проанализировать затраченные ресурсы)

Exit...................0 (Выйти из меню)

Проверка выражения может быть организована как массивом, так и списком.

Check expression by

Stack array....................1 (Стек-массив)

Stack list.......................2 (Стек-список)

Input string (Для проверки могут быть введены любые символы)

Работа со стеком может быть также реализована на массиве и на списке.

Exit...................0 (Выход из меню)

Push arguments.........1 (Добавление элементов любого типа в стек)

Pop arguments..........2 (Удаление элементов из стека)

Print stack............3 (Вывод содержимого стека на экран)

Для анализа ресурсов пользователь должен ввести количество добавляемых/удаляемых элементов.

Программист должен рассмотреть всевозможный формат ввода данных, ошибки, связанные с ошибочным вводом, ошибочным выделением памяти, обращением к пустому стеку и его переполнение. Также необходимо обработать ситуацию, когда пользователь пытается поместить в стек больше аргументов, чем ожидается.

*Выходные данные*

Результат проверки строки (корректна/некорректна).

Вывод на экран содержимого стека или адресов освобождаемых элементов.

Вывод затраченного времени и памяти для обоих вариантов реализации.

*Структура данных*

*1. Массив*

char arr\_stack[N];

char \*p\_curr = arr\_stack — 1;

*2.Список*

struct node

{

char value;

struct node \*previous\_el;

};

struct node \*last\_element = NULL;

***Описание алгоритма***

* Случай проверки

1. Считать строку с клавиатуры.

2. Пройти посимвольно всю строку

3. Если встретилась открывающаяся скобка — поместить ее в стек.

Если встретилась закрывающаяся скобка — сравнить с верхним элементом стеком. В случае совпадения удалить элемент из стека. Иначе — ввод некорректен.

* Случай работы со стеком.

1. push() - смещаем указатель к следующей ячейке памяти, записываем в стек текущий символ, увеличиваем количество элементов стека.

2. pop() - считываем данные, перемещаем указатель к предыдущему элементу, уменьшаем количество элементов стека.

***Тесты***

1. Попытка освободить большее количество элементов, чем есть в стеке.

Stack is empty.

2. Переполнение стека.

Overflow of stack.

***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 (LIFO).

2. Каким образом и сколько памяти выделяется под хранение стека при

различной его реализации?

3. Каким образом освобождается память при удалении элемента стека при

различной реализации стека?

|  |  |
| --- | --- |
| Массив | Список |
| Для хранения отводится непрерывная область памяти, ограниченная сверху и снизу. | Для хранения отводится область память, ограниченная сверху лишь доступным объемом ОП |
| При добавлении элемента указатель смещается к следующему элементу массива, затем записывается по этому адресу. | При добавлении элемента происходит выделение памяти, ее адрес записывается по указателю стека, затем по указателю помещается значение. |
| При удалении элемента считывается значение, а потом указатель смещается к предыдущему элементу. | Удаление элемента происходит аналогично массиву. Но память, выделенная под удаляемый элемент, освобождается. |

4. Что происходит с элементами стека при его просмотре?

В классической реализации стека просмотр содержимого без извлечения его элементов невозможен.

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

Стек эффективнее реализовывать списком как по памяти, так и по времени. Поскольку, каждый раз добавляя элемент в список, нам приходится сдвигать указатели, что затратно по времени. К тому же, в случае списка, мы должны выделять память под указатель на следующий элемент, что заметно отнимает память.

***Вывод***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Время (сек) | | Память (байт) | |
| Массив | Список | Массив | Список |
| 50 | 0.18  0.56 | | 58  808 | |
| 100 | 0.08  0.41 | | 108  1608 | |
| 200 | 0.055  0.29 | | 208  3208 | |
| 500 | 0.032  0.15 | | 508  8008 | |
| 1000 | 0.033  0.14 | | 1008  16008 | |

Мы научились организовывать стек в виде массива и в виде списка и, проанализировав данные, заметили, что массив работает эффективней как по памяти, так и по времени.

По времени массив работает на 23% быстрее, а по памяти на 60% эффективней. Такая разница в памяти связана с тем, что в случае списка мы храним значение (1 байт), указатель (8 байт) и у нас происходит выравнивание до 12 байт. То есть, 3 байта у каждого элемента остаются пустыми. А в случае массива, по байту для каждого элемента.