|  |  |
| --- | --- |
| **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

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4 «Работа со стеком» (ВАР. 9)**

Студент Смирнов Иван Владимирович

Группа ИУ7 – 32Б

Проверил Барышникова Марина Юрьевна

*2023 г.*

Оглавление

[Описание условия задачи 3](#_Toc149336720)

[Описание технического задания 3](#_Toc149336721)

[Описание внутренних структур данных 5](#_Toc149336722)

[Описание алгоритмов 6](#_Toc149336723)

[Сравнение времени работы и ресурсных затрат 6](#_Toc149336724)

[Тесты 8](#_Toc149336725)

[Ответы на контрольные вопросы 11](#_Toc149336726)

[Вывод 12](#_Toc149336727)

# Описание условия задачи

Программа работает со стеком, выполняет операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) статическим массивом (дополнительно можно реализовать динамическим 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. Так же в программе реализован перевод выражения в постфиксную форму с учетом приоритета выполнения операций. Вызов всех операций программы осуществлен с помощью интерактивного меню.

Описание технического задания

1. **Исходные данные и результат**

**Входные данные:**

Пользователь выбирает пункты меню с соответствующими действиями.

|  |
| --- |
| Меню:  1) Добавить элемент в стек (массив)  2) Удалить элемент из стека (массив)  3) Вывод текущего состояния стека (массив)  4) Ввести выражение и записать в стек (массив)  5) Вывести выражение в постфиксной форме (массив)  6) Вывести адреса памяти элементов стека (массив)  7) Вывести свободные адреса памяти стека (массив)  8) Добавить элемент в стек (список)  9) Удалить элемент из стека (список)  10) Вывод текущего состояния стека (список)  11) Ввести выражение и записать в стек (список)  12) Вывести выражение в постфиксной форме (список)  13) Вывести адреса памяти элементов стека (список)  14) Вывести свободные адреса памяти стека (список)  15) Сравнить время и ресурсы использования стека (массив) и стека (список)  16) Выход |

**Выходные данные:**

На экран выводятся результаты команд меню.

1. **Задачи, реализуемой программой**

**Программа требует от пользователя выбор типа работы со стеком (с помощью статического массива или с помощью списка). Программа умеет добавлять/удалять элемент из стека, выводить текущее состояние стека, записывать выражение с операциями и скобками в стек и затем выводить его в постфиксной форме. Также можно посмотреть адреса элементов стека и адреса освобожденных областей.**

**Способ обращения к программе**

**Программа запускается через терминал.**

1. **Возможные аварийные ситуации и ошибки пользователя**
2. **Пользователь ввел несуществующий номер пункта меню**

**На выходе сообщение: «Invalid choice. Please try again.»**

1. **Пользователь попытался ввести не целое число в стек**

**На выходе сообщение: «Элемент введен неправильно!»**

1. **Пользователь попытался добавить элемент в полный стек**

**На выходе сообщение: «Стек полон! (нельзя добавить элемент)»**

1. **Пользователь попытался удалить элемент из пустого стека**

**На выходе сообщение: «Стек пуст! (нельзя удалить элемент)»**

1. **Пользователь ввел некорректное выражение**

**На выходе сообщение: «Некорректное выражение!»**

# **Описание внутренних структур данных**

Стек в виде стат. массива и в виде списка хранятся в следующих структурах:

|  |
| --- |
| #define MAX\_SIZE 1000  typedef struct  {  int data[MAX\_SIZE];  int top;  } ArrayStack;  typedef struct Node  {  int data;  struct Node \*next;  } ListNode;  typedef struct  {  ListNode\* top;  } ListStack; |

ArrayStack – стек в виде статического массива, где data – данные стека, top – верхний элемент стека. ListStack – стек в виде списка, в котором хранится указатель на голову типа ListNode. В ListNode есть два поля: data – значение элемента списка, next – указатель на следующий элемент.

# Описание алгоритмов

* 1. После запуска программы пользователь выбирает пункт меню
  2. При добавлении элемента в стек (в виде списка) из массива свободных адресов убирается соответствующий адрес для элемента
  3. При удалении элемента из стека (в виде списка) в массив свободных адресов добавляется адрес освободившегося элемента
  4. При вводе выражения, запоминается его инфиксная форма и на основе стека создается постфиксная форма
  5. Адреса свободной памяти стека, представленного в виде статического массива, хранятся как не занятые адреса памяти
  6. Адреса свободной памяти стека, представленного в виде списка, хранятся как адреса памяти из под освободившихся элементов (которые подверглись pop-у)

# Сравнение времени работы и ресурсных затрат

В программе реализована функция, которая выполняет замерный эксперимент для стеков разных реализаций по функциям добавления/удаления элемента. Эксперимент проводился для 10, 50, 100, 500, 1000 операций. Замерялось время добавления элемента, удаление элемента и обоих операций вместе, а также количество байт, занимаемое в памяти стеками. Для каждого размера стека замеры проводились 100 раз, в качестве времени работы операций выдается среднее значение из 100 замеров. Информация выводится в следующем виде:

|  |
| --- |
| Замерный эксперимент сравнения stack\_array и stack\_list:  РАЗМЕР СТЕКА: 10  Статический массив:  Время (push X10): 0.24000000мкс.  Время (pop X10): 0.19000000мкс.  Время (push+pop X10): 0.94000000мкс.  Занимаемая память: 44Б  Список:  Время (push X10): 0.24000000мкс.  Время (pop X10): 0.28000000мкс.  Время (push+pop X10): 1.05000000мкс.  Занимаемая память: 200Б  Эффективность массива относительно списка:  Время (push X10): 0.0000%  Время (pop X10): 47.3684%  Время (push+pop X10): 11.7021%  РАЗМЕР СТЕКА: 50  Статический массив:  Время (push X50): 0.40000000мкс.  Время (pop X50): 0.35000000мкс.  Время (push+pop X50): 1.28000000мкс.  Занимаемая память: 204Б  Список:  Время (push X50): 0.69000000мкс.  Время (pop X50): 0.81000000мкс.  Время (push+pop X50): 2.04000000мкс.  Занимаемая память: 1000Б  Эффективность массива относительно списка:  Время (push X50): 72.5000%  Время (pop X50): 131.4286%  Время (push+pop X50): 59.3750%  РАЗМЕР СТЕКА: 100  Статический массив:  Время (push X100): 0.61000000мкс.  Время (pop X100): 0.57000000мкс.  Время (push+pop X100): 1.67000000мкс.  Занимаемая память: 404Б  Список:  Время (push X100): 0.99000000мкс.  Время (pop X100): 1.31000000мкс.  Время (push+pop X100): 2.79000000мкс.  Занимаемая память: 2000Б  Эффективность массива относительно списка:  Время (push X100): 62.2951%  Время (pop X100): 129.8246%  Время (push+pop X100): 67.0659%  РАЗМЕР СТЕКА: 500  Статический массив:  Время (push X500): 2.38000000мкс.  Время (pop X500): 2.14000000мкс.  Время (push+pop X500): 5.11000000мкс.  Занимаемая память: 2004Б  Список:  Время (push X500): 4.37000000мкс.  Время (pop X500): 6.71000000мкс.  Время (push+pop X500): 11.91000000мкс.  Занимаемая память: 10000Б  Эффективность массива относительно списка:  Время (push X500): 83.6134%  Время (pop X500): 213.5514%  Время (push+pop X500): 133.0724%  РАЗМЕР СТЕКА: 1000  Статический массив:  Время (push X1000): 4.49000000мкс.  Время (pop X1000): 3.61000000мкс.  Время (push+pop X1000): 8.67000000мкс.  Занимаемая память: 4004Б  Список:  Время (push X1000): 8.30000000мкс.  Время (pop X1000): 11.72000000мкс.  Время (push+pop X1000): 20.57000000мкс.  Занимаемая память: 20000Б  Эффективность массива относительно списка:  Время (push X1000): 84.8552%  Время (pop X1000): 224.6537%  Время (push+pop X1000): 137.2549% |

Из замерного эксперимента видно, что все операции при всех тестируемых размерах сработали по времени быстрее у статического массива, нежели чем у списка. По памяти также статический массив занимает меньше памяти, чем список. Чем больше размер стека, тем больше процент эффективности статического массива относительно списка. То есть для реализации стека лучше использовать статический массив, нежели список, а конкретно в тех случаях, когда память не требуется выделять динамически и освобождать.

# Тесты

В следующих тестах символ «>» обозначает ввод пользователем пункта меню.

1. **Позитивные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| pos\_01 | >1  1  >1  2  >3 | 1 2 | Умеет ли программа добавлять элементы в стек и выводить его |
| pos\_02 | >1  1  >1  2  >2  >3 | 1 | Умеет ли программа правильно удалять элементы из стека |
| pos\_03 | >4  2+3\*2  >5 | 2 3 2 \*+ | Умеет ли программа записывать выражение в постфиксную форму с учетом приоритета операций |
| pos\_04 | >4  (2+3)\*(4+5)  >5 | 2 3 + 4 5 + \* | Умеет ли программа обрабатывать выражение со скобками |
| pos\_05 | >1  1  >1 2  >1  3  >6 | Адреса памяти: 0x7fff1069a190 0x7fff1069a194 0x7fff1069a198 | Умеет ли программа выводить адреса памяти элементов стека |
| pos\_06 | >8  1  >8  2  >8 3  >13  >9  >13 >14 | Адреса памяти: 0x5605d1731e40 0x5605d17303a0 0x5605d1731c80  Адреса памяти: 0x5605d17303a0 0x5605d1731c80  Свободные адреса памяти: 0x5605d1731e40 | Умеет ли программа хранить и выводить информацию о свободных адресах памяти для стека в виде списка |

1. **Негативные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| neg\_01 |  | Сообщение: «Invalid choice. Please try again.» | Пустой ввод |
| neg\_02 | >-1 | Сообщение: «Invalid choice. Please try again.» | Правильность ввода пункта меню |
| neg\_03 | >100 | Сообщение: «Invalid choice. Please try again.» | Правильность ввода пункта меню |
| neg\_04 | >1  (пустой ввод или введена не цифра) | Сообщение: «Элемент введен неправильно!» | Некорректный ввод элемента |
| neg\_05 | >1  2 | Сообщение: «Стек полон! (нельзя добавить элемент)» | Попытка записать элемент в заполненный стек |
| neg\_06 | >2 | Сообщение: «Стек пуст! (нельзя удалить элемент)» | Попытка удалить элемент из пустого стека |
| neg\_07 | >4  ((9+0) | Сообщение:  «Некорректное выражение!» | Правильность ввода выражения (кол-во ) |

# Ответы на контрольные вопросы

1. **Что такое стек?**

Стек - это структура данных, которая работает по принципу "последний вошел, первый вышел" (LIFO). Он представляет собой контейнер, в котором элементы добавляются и удаляются только с одного конца, называемого вершиной стека.

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

Различные реализации стека могут использовать разное количество памяти. В общем случае, при реализации стека на основе массива, выделяется фиксированное количество памяти, равное размеру стека. При реализации стека на основе связного списка, память выделяется динамически при добавлении новых элементов.

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

При удалении элемента стека в различных реализациях, освобождение памяти может происходить по-разному. В случае стека на основе динамического массива, память не освобождается явно, так как просто перемещается указатель вершины стека. В случае статического массива память из под стека не освободится до завершения программы. В случае стека на основе связного списка, память, занимаемая удаленным элементом, освобождается с помощью операции освобождения памяти.

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

При просмотре стека и вывода его элементов, сами элементы убираются из стека и в итоге стек становится пустым.

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

Эффективность реализации стека зависит от конкретных требований и ограничений. Если требуется фиксированный размер стека и известно максимальное количество элементов, то реализация на основе массива может быть эффективнее, так как не требует дополнительной памяти для хранения связей между элементами. Однако, если размер стека может изменяться динамически, то реализация на основе связного списка может быть более гибкой и эффективной, так как позволяет динамически выделять и освобождать память под элементы стека.

# Вывод

В рамках задания были реализованы разные представления стека (в виде статического массива и связанного списка), операции добавления/удаления/вывода стека на экран. В программе предусмотрена команда для вывода адресов памяти элементов стека, а также вывод свободных ячеек памяти. При этом при добавлении нового элемента в стек, представленный в виде списка, задействуются освобожденные ранее участки памяти, соответственно данные действия не будут приводить к фрагментации. Выражение, заданное числами, операциями и скобками переводится в постфиксную форму с помощью стека.