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 **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04** Программная инженерия

О Т Ч Е Т

По лабораторной работе № 4

Название: Работа со стеком

Дисциплина: Типы и структуры данных

Студент ИУ7-32Б Беляк С.С.

Преподаватель Барышникова М.Ю.

Москва, 2023

Описание условия задачи

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) статическим 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. Ввести арифметическое выражение типа: число |знак| … число |знак| число. Вычислить значение выражения. (Приоритетность операций необязательна)

Описание задачи, реализуемой в программе

Цель работы - реализовать операции работы со стеком, который представлен в виде статического массива и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации; получить представление о механизмах выделения и освобождения памяти при работе со стеком

Описание ТЗ.

Описание исходных данных

Пользователь выбирает опцию из меню, представленного в консоли.

Возможные варианты выбора:

Меню:

1. Вычислить арифметическое выражение (с использованием массива)
2. Вычислить арифметическое выражение (с использованием списка)
3. Добавить элемент в стек (с использованием массива)
4. Удалить элемент из стека (с использованием массива)
5. Вывести стек (с использованием массива)
6. Добавить элемент в стек (с использованием списка)
7. Удалить элемент из стека (с использованием списка)
8. Вывести стек (с использованием списка)
9. Сравнить производительность
10. Вывести массив свободных элементов

0. Выйти из меню

Пользователь может выбрать соответствующую опцию, введя число от 0 до 10 в консоли.

Имеются следующие ограничения:

Ограничение по формату ввода данных:

Программа ожидает ввод данных в формате целых чисел. Это означает, что пользователь должен вводить только целочисленные значения при выполнении операций, связанных с добавлением элементов в стек и арифметическими вычислениями. Ввод данных в других форматах может привести к некорректным результатам или ошибкам выполнения программы.

Ограничение по размеру стека:

Программа может иметь ограничение на максимальный размер стека, как это определено в реализации стека (в случае стека на основе массива). Пользователь должен быть осторожен при добавлении элементов в стек, чтобы избежать переполнения стека и потенциальных ошибок программы.

Ограничение по максимальной длине арифметических выражений:

Для операции вычисления арифметических выражений, программа может иметь ограничение на максимальную длину введенных выражений. Это ограничение может быть связано с размером буфера для ввода данных. Если введенное выражение слишком длинное, программа может выдавать ошибку.

Ограничение по возможным операциям:
Некоторые операции могут быть ограничены и доступны только при определенных условиях. Например, удаление элемента из пустого стека может быть недопустимой операцией.

2. Описание результатов программы

1. Вычислить арифметическое выражение (с использованием массива):
    - Программа вычисляет значение этого выражения с использованием стека на основе статического массива.
    - Результат вычисления выводится в консоль.
2. Вычислить арифметическое выражение (с использованием списка):
    - Программа вычисляет значение этого выражения с использованием стека на основе связанного списка.
    - Результат вычисления выводится в консоль.
3. Добавить элемент в стек (с использованием массива):
    - Число добавляется в стек, реализованный с использованием статического массива.
    - Программа сообщает, что элемент был успешно добавлен.
4. Удалить элемент из стека (с использованием массива):
    - Последний элемент из стека, реализованного с использованием статического массива, удаляется.
    - Программа сообщает, что элемент был успешно удален.
5. Вывести стек (с использованием массива):
    - Текущее состояние стека, реализованного с использованием статического массива, выводится в консоль.
6. Добавить элемент в стек с использованием списка):
    - Число добавляется в стек, реализованный с использованием связанного списка.
    - Программа сообщает, что элемент был успешно добавлен.
7. Удалить элемент из стека (с использованием списка):
    - Последний элемент из стека, реализованного с использованием связанного списка, удаляется.
    - Программа сообщает, что элемент был успешно удален.
8. Вывести стек (с использованием списка):
    - Текущее состояние стека, реализованного с использованием связанного списка, выводится в консоль.
9. Сравнить производительность:
    - Программа проводит серию экспериментов для сравнения производительности двух реализаций стека: массива и списка.
    - Результаты сравнения включают в себя информацию о времени выполнения операций и использовании памяти для разных операций и размеров стека.
    - Результаты выводятся в консоль.
10. Вывести массив свободных областей
    - Вывод массива освобожденных участков памяти в консоль
0. Выход из программы:
    - Программа завершает выполнение

3. Способ обращения к программе

Способ обращения к программе пользователем происходит через исполняемый файл app.exe.

4. Описание возможных аварийных ситуаций и ошибок пользователя

CHOICE_ERROR (Ошибка выбора)
Эта ошибка возникает, когда пользователь вводит некорректное значение в меню программы. Например, если пользователь вводит символ или значение, которое не соответствует доступным опциям в меню.
При возникновении этой ошибки программа может завершиться с кодом ошибки. Пользователю будет предложено ввести корректное значение из доступных опций.

ALLOCATE_ERROR (Ошибка выделения памяти)
Эта ошибка возникает, когда программа не может выделить дополнительную динамическую память для добавления новых элементов в стек, либо для создания новых узлов в связанном списке. При возникновении этой ошибки, программа может завершиться с кодом ошибки. Это может произойти, если доступной памяти не хватает для выполнения операции. Программа должна обработать эту ошибку и сообщить пользователю о невозможности выполнения операции.

STACK_OVERFLOW_ERROR (Ошибка переполнения стека)
Эта ошибка возникает, когда стек переполняется, то есть при попытке добавления элемента в стек, уже достигнут максимально допустимый размер стека.
При возникновении этой ошибки, программа может завершить выполнение с кодом ошибки. Это означает, что пользователь попытался добавить элемент в полный стек, и операция не может быть выполнена.

EMPTY_STACK_ERROR (Ошибка при попытке удаления элемента из пустого стека)
Эта ошибка возникает, когда пользователь пытается удалить элемент из стека, но стек уже пуст.
При возникновении этой ошибки, программа может завершить выполнение с кодом ошибки. Это означает, что операция удаления элемента из пустого стека не имеет смысла и не может быть выполнена.

5.Описание внутренних структур данных

Структуры в программе служат для представления и управления стеком на основе массива и стеком на основе списка:

```
/**
 * @struct Node
 * @brief Структура для представления узла в стеке на основе списка.
 */
typedef struct Node {
    int value;          /**< Значение элемента в узле. */
    struct Node *next;  /**< Указатель на следующий узел. */
} Node;
typedef struct {
    Node *Stack_Pointer; /**< Указатель на вершину стека. */
    int curr_size;       /**< Текущий размер стека. */
    int max_size;        /**< Максимальный размер стека. */
} List_Stack;
typedef struct FreeMemoryBlock {
    void* address;
    size_t size;
    struct FreeMemoryBlock* next;
} FreeMemoryB
```

1. **Структура Node (Узел в списке):**

Структура представляет элемент (узел) в стеке на основе списка.

Содержит два поля:

- **value**: Хранит значение элемента в узле (целое число).
- **next**: Указатель на следующий узел в списке.

2. **Структура List_Stack (Стек на основе списка):**

Структура используется для представления стека на основе списка.

Содержит следующие поля:

- **Stack_Pointer**: Указатель на вершину стека (на последний добавленный элемент).
- **curr_size**: Текущий размер стека (количество элементов в стеке).
- **max_size**: Максимальный размер стека (максимальное количество элементов, которое может хранить стек).

3. **Структура FreeMemoryBlock (Свободные области памяти):**

Эта структура используется для отслеживания свободных областей памяти в программе.

Содержит следующие поля:

- **address**: Указатель на начало свободной области памяти.
- **size**: Размер свободной области памяти.
- **next**: Указатель на следующую свободную область памяти.

4. **Структура Arr_Stack (Стек на основе массива):**

Структура используется для представления стека на основе массива.

```
/**
 * @struct Arr_Stack
 * @brief Структура для представления стека на основе массива.
 */
typedef struct {
    int Stack_Start[STACK_SIZE];
    int curr_size;
} Arr_Stack;
```

Содержит следующие поля:

- **Stack_Start**: Массив, представляющий стек.
- **curr_size**: Текущий размер стека (количество элементов в стеке).

Эти структуры и функции служат для представления и управления стеком в программе, как на основе списка, так и на основе массива. Они обеспечивают функциональность добавления элементов, удаления элементов и другие операции, связанные со стеком.

1. Алгоритм программы

Общий алгоритм программы:

1. Инициализация системы и выделение памяти:
2. Проверка выделения памяти:
3. После выделения памяти, программа проверяет успешность этой операции. Если выделение памяти завершилось неудачей, программа может выдать ошибку и завершить работу.
4. Основной цикл программы:
5. Программа входит в главный цикл, который ожидает ввода пользователя и предоставляет меню с различными опциями.
6. В зависимости от выбора пользователя, программа выполняет следующие действия:

- Ввод арифметического выражения или элемента для добавления в стек (с использованием массива или списка).
- Вычисление арифметического выражения (с использованием массива или списка).
- Добавление элемента в стек (с использованием массива или списка).
- Удаление элемента из стека (с использованием массива или списка).
- Вывод состояния стека (с использованием массива или списка).
- Сравнение производительности двух реализаций стека (эксперименты).
- Вывод массива свободных областей
- Выход из программы.

7. Возврат в главное меню:
8. При завершении программы, программа освобождает выделенную память для структур данных, завершает работу и закрывает все ресурсы.

Функция **calculate_list_stack**

1. Создается стек с использованием списка.
2. Происходит разбор арифметического выражения.
3. При нахождении числа, оно добавляется в стек.
4. При нахождении оператора, он применяется к двум верхним элементам стека.
5. Вычисленное значение также добавляется в стек.
6. В конце выполнения арифметического выражения извлекается результат и возвращается.

Функция **calculate_arr_stack**

1. Создается стек с использованием массива.
2. Происходит разбор арифметического выражения.
3. При нахождении числа, оно добавляется в стек.
4. При нахождении оператора, он применяется к двум верхним элементам стека.
5. Вычисленное значение также добавляется в стек.
6. В конце выполнения арифметического выражения извлекается результат и возвращается.

Функция **push_arr_stack**

1. Проверяется, что стек не достиг своего максимального размера (иначе возвращается ошибка).
2. Значение добавляется в массив стека.
3. Размер стека увеличивается на 1.

Функция **push_list_stack**

1. Создается новый узел с указанным значением.
2. Проверяется успешное выделение памяти для узла.
3. Если стек полон (достиг своего максимального размера), увеличивается его размер.
4. Новый узел становится вершиной стека.
5. Размер стека увеличивается на 1.

Функция **pop_list_stack**

1. Проверяется, что стек не пуст (иначе возвращается ошибка).
2. Верхний элемент стека извлекается.
3. Удаляется узел, соответствующий этому элементу.
4. Размер стека уменьшается на 1.
5. Извлеченное значение возвращается.

Функция **pop_arr_stack**

1. Проверяется, что стек не пуст (иначе возвращается ошибка).
2. Верхний элемент стека извлекается и возвращается.
3. Размер стека уменьшается на 1.

Функция **free_list_stack**

1. Освобождается память, занимаемая узлами стека.
2. Затем освобождается память, занимаемая самим стеком.

Функция **free_arr_stack**

1. Освобождается память, занимаемая стеком.

Функция **print_list_stack**

1. Выводится содержимое стека на основе списка, включая значения элементов и адреса

Функция **print_arr_stack**

1. Выводится содержимое стека на основе массива, включая значения элементов и адреса.

Функция **is_operator**

1. Проверяется, является ли символ оператором ('+', '-', '*', '/').

Функция **perform_operation**

1. Выполняется арифметическая операция между двумя операндами в соответствии с заданным оператором.

Функция **run_experiments**

1. Запускается набор экспериментов для сравнения производительности стеков на основе массива и списка.
2. Для каждой операции и размера стека производятся повторные эксперименты.
3.Измеряется время выполнения и использование памяти для каждой операции и размера стека.
4. Выводятся результаты в виде таблицы сравнения производительности.

Вывод по алгоритму работающей программы:

Алгоритм программы предоставляет удобный способ управления стеками и позволяет выполнять арифметические операции, а также предоставляет информацию о производительности стеков на основе разных структур данных. Программа предоставляет удобный интерфейс для работы со стеками и позволяет оценить эффективность алгоритмов.

Тесты

Таблица положительных тестов.

Номер теста	Описание	Ожидаемый результат
1	Вычисление выражения "10 - 2 / 2" с использованием стека на основе списка.	Результат вычисления равен 4.
2	Вычисление выражения "5 + 3 * 7" с использованием стека на основе массива.	Результат вычисления равен 56.
3	Добавление элемента в стек на основе массива, начальный размер 0, элемент 42.	Стек на основе массива содержит элемент 42, размер увеличился на 1.
4	Добавление элемента в стек на основе списка, начальный размер 0, элемент 99.	Стек на основе списка содержит элемент 99, размер увеличился на 1.
5	Удаление элемента (массив)	Стек на основе массива содержит один элемент меньше.
6	Удаление элемента (список)	Стек на основе списка содержит один элемент меньше.
9	Сравнение эффективности программы для стека на основе массива и стека на основе списка	Вывод результатов сравнения.

Таблица негативных тестов.

Номер теста	Описание	Ожидаемый результат
1	Попытка удаления элемента из пустого стека на основе списка	Ошибка: пустой стек
2	Попытка удаления элемента из пустого стека на основе массива	Ошибка: пустой стек
3	Попытка выделения памяти, когда нет свободной памяти (массив)	Ошибка: переполнение стека
4	Попытка выделения памяти, когда нет свободной памяти (список)	Ошибка: выделение памяти
5	Введение недопустимого символа в выражение (например, буквы)	Ошибка: недопустимый символ
6	Попытка деления на ноль в арифметическом выражении	Ошибка: деление на ноль
7	Превышение максимального размера стека (при попытке добавления элемента) (массив)	Ошибка: переполнение стека
8	Превышение максимального размера стека (при попытке добавления элемента) (список)	Ошибка: выделение памяти
9	Ввод некорректных значений для выбора в меню	Ошибка: Некорректный выбор в меню.

Временная эффективность и затраты памяти

Проанализируем результаты сравнения производительности стеков на основе массива и связанного списка. В таблице представлены результаты для четырех арифметических операций: сложения (+), вычитания (-), умножения (*), и деления (/), а также для четырех разных размеров стека: 10, 50, 100 и 1000 элементов. Данное вычисление было получено по среднему вычислению значения среди 10000 итераций.

Операция	Размер Stack	Память Array, байт	Память List, байт	Время (наносекунды) Array	Время (наносекунды) List
+	10	560000	580000	155.588900	423.371600
+	50	2960000	2980000	2654.938700	4300.549300
+	100	5960000	5980000	2746.348600	3108.098300
+	1000	45010000	45010000	4438.028700	10445.138700
-	10	560000	580000	148.659300	1330.888400
-	50	2960000	2980000	746.942900	8368.566800
-	100	5960000	5980000	3440.572600	7170.950500
-	1000	45010000	45010000	2986.412900	15554.037700
*	10	560000	580000	107.071400	295.004000
*	50	2960000	2980000	512.562400	3983.414600
*	100	5960000	5980000	3117.950900	5540.342600
*	1000	45010000	45010000	5035.293000	8189.002600
/	10	560000	580000	581.202600	1128.721700
/	50	2960000	2980000	4961.801600	5305.418100
/	100	5960000	5980000	4315.573300	3214.213200
/	1000	45010000	45010000	4135.512000	8657.686400

Эффективность работы со стеком на основе массива по операциям:

1. Сложение (+):

Количество элементов	Во сколько раз массива быстрее списка
10	В 2.72 раза
50	В 1.62 раза
100	В 1.13 раза
1000	В 2.36 раза

2. Вычитание (-):

Количество элементов	Во сколько раз массива быстрее списка
10	В 8.95 раза
50	В 11.19 раза
100	В 2.09 раза
1000	В 5.21 раза

3. Умножение (*):

Количество элементов	Во сколько раз массива быстрее списка
10	В 2.76 раза
50	В 7.78 раза
100	В 1.78 раза
1000	В 1.63 раза

4. Деление (/):

Количество элементов	Во сколько раз массива быстрее списка
10	В 1.94 раза
50	В 1.07 раза
100	В 0.74 раза
1000	В 2.09 раза

Эффективность работы со стеком на основе массива по размеру:

Количество элементов	Во сколько раз массива быстрее списка
10	В 4,09 раза
50	В 5,41 раза
100	В 1,43 раза
1000	В 2,82 раза

Выводы по эффективности:

Размер стека:

Видно, что время выполнения операций в стеке на основе массива остается сравнительно невысоким даже при больших размерах стека. По обобщенным данным, массив в среднем быстрее связанного списка в 3.19 раза.

Память:

Оба типа стеков требуют памяти для хранения элементов. Память для массива и связанного списка в целом увеличивается пропорционально размеру стека. Однако стек на основе массива в большинстве случаев более эффективен с точки зрения использования памяти. В среднем массив занимает память на 1.01 раза меньше по сравнению со списком.

Время:

Массив оказывается более эффективным в выполнении всех операций, показывая преимущество от 1.07 до 11.19 раз. Различия в эффективности времени выполнения операций становятся более значительными при увеличении размера данных. Среднее соотношение времени выполнения операций массива к списку составляет примерно 3.32 раза быстрее.

Итог:

Стек на основе массива чаще всего более эффективен с точки зрения времени выполнения и использования памяти, особенно при больших размерах стека.

Стек на основе связанного списка может иметь свои преимущества в определенных ситуациях, например, когда размер стека может динамически изменяться.

Вывод

В данной лабораторной работе было проведено равнение производительности стеков на основе массива и связанного списка. Были рассмотрены четыре арифметические операции: сложение, вычитание, умножение и деление, а также четыре разных размера стека: 10, 50, 100 и 1000 элементов.

Результаты сравнения были представлены в виде таблицы, в которой указаны размер стека, объем занимаемой памяти для массива и связанного списка, а также время выполнения операций для обоих типов стеков. В результате анализа, мы видим, что стек на основе массива показывает лучшую эффективность во всех случаях.

Время выполнения операций в стеке на основе массива остается сравнительно невысоким даже при больших размерах стека, стек на основе массива демонстрирует более высокую эффективность как по времени выполнения операций, так и по использованию памяти.

В среднем, массив оказывается быстрее связанного списка в 3.32 раза. Таким образом, стек на основе массива представляет собой предпочтительный выбор, особенно при работе с большими объемами данных.

Оба типа стеков, основанных на массиве и связанном списке, требуют памяти для хранения элементов. В целом, память для массива и связанного списка увеличивается пропорционально размеру стека. Однако стек на основе массива в большинстве случаев более эффективен с точки зрения использования памяти, занимая в среднем на 1.01 раза меньше места по сравнению со связанным списком.

В итоге, стек на основе массива чаще всего более эффективен с точки зрения времени выполнения и использования памяти, особенно при больших размерах стека.

Стек на основе связанного списка может иметь свои преимущества в определенных ситуациях, например, когда размер стека может динамически изменяться.

Ответы на контрольные вопросы

Что такое стек?

Стек – это последовательный список с переменной длиной, в котором включение исключение элементов происходит только с одной стороны – с его вершины. Стек функционирует по принципу: последним пришел – первым ушел, Last In – First Out (LIFO).

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

Реализации стека могут быть разными, и количество выделяемой памяти зависит от конкретной реализации. Вот несколько примеров:

Стек на основе массива: Память выделяется под массив фиксированного размера, который используется для хранения элементов стека. Размер массива определяется заранее, и вся память для стека выделяется одновременно.

Стек на основе связанного списка: Здесь память выделяется динамически при добавлении каждого элемента. Каждый элемент (узел) стека содержит указатель на следующий элемент и значение. Таким образом, память выделяется по мере необходимости, и стек может расти динамически.

Количество памяти, выделенной для стека, зависит от его размера и структуры данных, используемых для его реализации.

Каким образом освобождается память при удалении элемента стека при различной реализации стека?

При удалении элемента из стека, освобождение памяти происходит по-разному в зависимости от реализации стека:

Стек на основе массива: При удалении элемента из массива, память не освобождается явным образом. Просто уменьшается указатель (индекс) на вершину стека. Это позволяет заменить старое значение новым при следующем добавлении элемента.

Стек на основе связанного списка: При удалении элемента из стека на основе связанного списка, удаляется соответствующий узел, и память, выделенная под этот узел, освобождается с помощью функции free().

Что происходит с элементами стека при его просмотре?

При просмотре элементов стека, они остаются в стеке в неизменном виде. Просто читается значение элемента с вершины стека, но этот элемент не удаляется. Таким образом, элементы стека не изменяются при его просмотре.

Каким образом эффективнее реализовывать стек? От чего это зависит?

Эффективность реализации стека зависит от конкретных требований и ограничений приложения:

Стек на основе массива более эффективен по памяти, если известен максимальный размер стека заранее. Он может работать быстрее, чем связанный список, при доступе к элементам по индексу. Однако его размер ограничен.

Стек на основе связанного списка более гибок и способен динамически расти и сжиматься в зависимости от потребности. Это подходит для случаев, когда размер стека заранее неизвестен или может изменяться.

Эффективность также зависит от операций, которые выполняются чаще: добавление, удаление или просмотр элементов, а также от общей структуры программы.