|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4  
«РАБОТА СО СТЕКОМ»**

Название предмета: Типы и структуры данных

Студент: Малышев Иван Алексеевич

Группа: ИУ7-31Б

*2020 г.*

**Цель работы**: реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) 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 условия задачи**

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека. Реализовать стек:

1. массивом;
2. 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.

Элементами стека являются адреса памяти. При реализации массивами - их вводить, при реализации списком – брать адрес выделенной памяти под элемент.

**Техническое задание**

**Входные данные**

1. **Целое число, представляющее собой номер команды:** целое число от 0 до 8.
2. **Командно-зависимые данные:** адрес памяти.

**Выходные данные**

1. Результат выполнения индивидуального задания – содержимое стеков в виде массива и списка, содержимое вектора свободных областей.
2. Количественная характеристика сравнения вариантов обработки стека.

**Функция программы:** программа выполняет ряд функций, указанных при её первом запуске. Она позволяет:

1. Добавить элемент в стек в виде массива
2. Удалить элемент из стека в виде массива
3. Вывести стек в виде массива
4. Добавить элемент в стек в виде списка
5. Удалить элемент из стека в виде списка
6. Вывести стек в виде списка
7. Вывести вектор свободных областей
8. Сравнить время работы и объём требуемой памяти стеков

**Способ обращения к программе:**

К программе можно обратиться через терминал ОС по её имени «app.exe». Дальнейшие инструкции будут выведены после запуска.

**Аварийные ситуации:**

* Некорректный номер операции, т. е. число меньше нуля или больше 8
* Ввод некорректного элемента в стек в виде массива
* Попытка добавить элемент при заполненном стеке
* Попытка удалить элемент из пустого стека
* Попытка вывести пустой стек

Во всех указанных случаях программа завершится корректно или сообщит об ошибке.

**Структуры данных**

Реализация стека на основе статического массива

typedef struct

{

void \*stack[MAXVAL]; // статический массив (Макс. размер - 1000 элементов)

long ps; // указатель на текущий элемент стека

long max\_size; // максимальный размер стека

} array\_stack\_t;

Реализация стека на основе линейного односвязного списка:

typedef struct list\_stack

{

struct list\_stack \*elem; // Указатель на текущий узел

struct list\_stack \*prev; // Указатель на предыдущий узел

} list\_stack\_t;

Реализация вектора свободных областей:

typedef struct

{

list\_stack\_t \*\*stack; // вектор свободных областей

long ps; // указатель на текущий элемент вектора

} trash\_can\_t;

**Функции**

*int push\_array\_stack(array\_stack\_t \*stack, void \*elem)* – добавление элемента в стек в виде массива

*int pop\_array\_stack(array\_stack\_t \*stack)* – удаление элемента из стека в виде массива

*void print\_array\_stack(array\_stack\_t \*stack)* – печать стека в виде массива

*list\_stack\_t \*push\_list\_stack(list\_stack\_t \*stack)* - добавление элемента в стек в виде списка

*list\_stack\_t \*pop\_list\_stack(list\_stack\_t \*stack, trash\_can\_t \*trash\_can)* - удаление элемента из стека в виде списка

*void print\_list\_stack(list\_stack\_t \*stack)* - печать стека в виде списка (печатается в обратном порядке, т. е. на первом месте стоит последний элемент)

*void free\_list\_stack(list\_stack\_t \*stack)* – освобождение памяти из под стека в виде списка

*int push\_trash\_can(trash\_can\_t \*trash\_can, struct list\_stack \*elem\_ptr)* – добавление элемента в вектор свободных областей

*void print\_trash\_can(trash\_can\_t \*trash\_can)* – печать вектора свободных областей

*void free\_trash\_can(trash\_can\_t \*trash\_can)* – освобождение памяти из под вектора свободных областей

*void measuring(void)* – функция сравнения реализаций стеков

*uint64\_t tick(void)* – функция измерения времени в тактах процессора

**Алгоритм**

1. Пользователь вводит номер операции из меню.
2. Пока пользователь не введёт 0 (выход из программы), ему будет предложено выполнять действия со стеками.
3. При добавлении элемента в стек в виде массива, элемент массива вводится с клавиатуры; в стек в виде списка добавляется адрес текущего узла.
4. При удалении элемента из стека в виде списка, перед удалением он помещается в вектор свободных областей, в отличие от стека в виде массива.

**Тестирование**

Положительные тесты

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Добавление элемента в стек в виде массива при неполном стеке | 1  2345f | Элемент в стек в виде массива успешно добавлен |
| 2 | Добавление элемента в стек в виде списка при неполном стеке | 4 | Элемент в стек в виде списка успешно добавлен |
| 3 | Удаление элемента из стека в виде массива при непустом стеке | 2 | Элемент из стека в виде массива успешно удалён |
| 4 | Удаление элемента из стека в виде списка при непустом стеке | 5 | Элемент из стека в виде списка успешно удалён |
| 5 | Печать вектора свободных областей при непустом векторе | 7 | (Выводится содержимое вектора) |
| 6 | Печать стека в виде массива при непустом стеке | 3 | (Выводится содержимое стека в виде массива) |
| 7 | Печать стека в виде списка при непустом стеке | 6 | (Выводится содержимое стека в виде списка) |

Негативные тесты

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Добавление элемента в стек в виде массива при полном стеке | 1  2345f | Стек в виде массива переполнен |
| 2 | Добавление элемента в стек в виде списка при неполном стеке | 4 | Стек в виде списка переполнен |
| 3 | Удаление элемента из стека в виде массива при пустом стеке | 2 | Стек пуст |
| 4 | Удаление элемента из стека в виде списка при пустом стеке | 5 | Стек пуст |
| 5 | Печать вектора свободных областей при пустом векторе | 7 | Массив освободившихся адресов пуст |
| 6 | Печать стека в виде массива при пустом стеке | 3 | Стек пуст |
| 7 | Печать стека в виде списка при пустом стеке | 6 | Стек пуст |
| 8 | Ввод недопустимого номера команды | -1, 12 или А | Неизвестная команда! |
| 9 | Ввод недопустимого элемента стека в виде массива | gffgq | Ошибка ввода элемента |

**Оценка эффективности**

Измерения эффективности реализаций стека будут производиться в единицах измерения – тактах процессоров. Для измерения была специально написана ассемблерная функция, поэтому погрешность измерений минимальна. При записи результатов использовалось среднее количество тактов, полученное по результатам 100 измерений.

|  |  |  |  |
| --- | --- | --- | --- |
| Данные для стека в виде массива | | | |
| Длина | Время добавления | Время удаления | Объём памяти (в байтах) |
| 10 | 166 | 108 | 88 |
| 100 | 1578 | 1210 | 808 |
| 250 | 3991 | 2654 | 2008 |
| 500 | 7671 | 5132 | 4008 |
| 1000 | 15343 | 10356 | 8008 |

|  |  |  |  |
| --- | --- | --- | --- |
| Данные для стека в виде списка | | | |
| Длина | Время добавления | Время удаления | Объём памяти (в байтах) |
| 10 | 1465 | 4068 | 160 |
| 100 | 16470 | 40118 | 1600 |
| 250 | 44239 | 106888 | 4000 |
| 500 | 87209 | 219458 | 8000 |
| 1000 | 147627 | 379910 | 16000 |

**Контрольные вопросы**

**1. Что такое стек?**

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

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

При хранении стека с помощью списка, 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 достаётся верхний элемент стека.

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

Реализовывать стек эффективнее с помощью массива. Он выигрывает как во времени обработки, так и в количестве занимаемой памяти (в классическом случае). Вариант хранения списка может выигрывать только в том случае, если стек реализован статическим массивом. В этом случае, память для списка ограничена размером оперативной памяти (так как память выделяется в куче), а память для статического массива ограничена размером стека.

**Вывод**

Как видно из результатов оценки эффективности, реализация стека в виде списка проигрывает реализации стека в виде статического массива по всем параметрам. Во-первых, список требует почти в 2 раза больше памяти, чем массив. Во-вторых, добавление элемента в список медленнее добавления в массив примерно в 9-10 раз во всех случаях. В-третьих, удаление из списка занимает примерно в 40 раз больше тактов, чем удаление из массива. Такая разница объясняется тем, что во время удаления из списка удаляемый элемент помещается в вектор свободных областей, что тоже занимает много времени.

Таким образом, можно сделать вывод, что если в задаче требуется стек, размер которого в заранее известен или ограничен (не оперативной памятью), то лучше использовать реализацию в виде статического массива, т. к. он быстрее и занимает меньше памяти. В иных случая – целесообразнее использовать реализацию стека в виде списка.