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

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

Название предмета: Типы и структуры данных

Студент: Варламова Екатерина Алексеевнa

Группа: ИУ7-31Б

*2020г.*

1. **Описание условия задачи**

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) массивом; б) списком. Оценить преимущества и недостатки каждой реализации. 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. С помощью стека проверить правильность расстановки скобок трех типов (круглых, квадратных и фигурных) в выражении.

**Техническое задание**

1. ***Описание исходных данных***

Программа ожидает:

- название файла с выражением (в аргументах командной строки)

- максимальный размер стека

- номер действия (подробнее о том, какой номер соответствует определённому действию, будет выведено при запуске программы).

Формат ввода и ограничения:

1. При добавлении элемента в стек требуется ввести элемент, который имеет символьный тип (символы из однобайтовых кодировок).
2. При вводе максимального размера стека ожидается целое неотрицательное число.
3. ***Описание результата программы***

Результатом работы программы могут являться (в зависимости от введённого действия):

1. Извлечённый элемент стека.
2. Текущее состояние стека (если стек реализован списком, помимо самих элементов будут также выведены их адреса и список адресов всех извлечённых элементов).
3. Вывод о том, правильно ли расположены скобки в выражении. При этом при добавлении в стек и удалении элементов (скобок) из стека будет выводиться текущее состояние стека (текущее состояние определяется так же, как в предыдущем пункте).
4. Статистика по времени выполнения и объёму памяти при обработке стеков, реализованных списком и динамическим массивом.

Формат вывода:

1. Для извлечённого элемента:

extracted value: <element>

1. Для текущего состояния стека, если он реализован:

* массивом (в стеке 5 элементов):

<element> <element> <element> <element> <element>

* списком (в стеке 5 элементов, 2 были извлечены):

element: <element> located: <address>

element: <element> located: <address>

element: <element> located: <address>

element: <element> located: <address>

element: <element> located: <address>

free: <address> <address>

Пример:

element: e located: 0x7fab19604090

element: d located: 0x7fab194058a0

element: c located: 0x7fab19405890

element: b located: 0x7fab19604080

element: a located: 0x7fab195040b0

free: 0x7fab19604090 0x7fab194058a0

1. Для вывода о расположении скобок:

* correct –правильная расстановка скобок;
* incorrect – одинаковое количество открытых и закрытых скобок, но какая-то скобка закрыта неправильно. Пример: ({}];
* incorrect: stack overflow – при чтении выражения переполнился стек;
* incorrect: memory error – операционная система отказала в выделении памяти под узел списка или под область вектора;
* incorrect: not enough brackets – количество открытых и закрытых скобок в выражении не совпадает

1. Статистика выводится в виде таблицы.
2. ***Описание задачи, реализуемой программой***

Программа показывает работу стеков, реализованных на основе динамического массива и списка. Можно добавлять элемент в стек, извлекать из него элемент и просматривать текущее состояние стека.

Кроме того, программа может считывать выражение из файла (имя должно быть указано в аргументах командной строки) и определять, правильно ли расставлены скобки в выражении. Эта функция работает с обоими стеками: реализованного на основе списка и на основе массива. Сравнить эффективность стеков можно в разделе статистика.

1. ***Способ обращения к программе***

Способ обращения к программе - консольный. Дальнейшие инструкции будут выведены после запуска.

1. ***Описание возможных аварийных ситуаций и ошибок пользователя***

* ошибки ввода максимального размера стека и элементов стека;
* ошибка ввода действия;
* ошибка в названии файла или его отсутствие (физическое и/или в аргументах командной строки);
* отказ операционной системы выделить запрашиваемую память;

Во всех указанных случаях программа сообщит об ошибке.

1. **Описание внутренних структур данных**

В программе есть 2 основных структуры данных:

1. ***Стек на основе списка***

Описание на языке C стека выглядит таким образом:

typedef struct node

{

struct node \*next;

elem\_t value;

} node\_t;

typedef struct

{

node\_t \*head;

size\_t count;

size\_t max\_count;

size\_t size\_of\_element;

vector\_t adrs;

} stack\_on\_list\_t;

Данные хранятся в форме односвязного линейного списка. Структура стека состоит из: указателя head на голову списка (вершина стека), текущего количества элементов и максимального count и max\_count соответственно, размера элементов size\_of\_element. Дополнительно ввиду условия задачи хранится вектор adrs адресов освобождённых элементов списка. У данной структуры есть следующие особенности:

* данные в памяти располагаются в произвольном порядке;
* при добавлении элемента каждый раз запрашивается память под элемент;
* при удалении элемента каждый раз память освобождается;

1. ***Стек на основе динамического массива***

Описание на языке C стека выглядит таким образом:

typedef struct {

vector\_t data;

elem\_t \*head;

size\_t count;

size\_t max\_count;

size\_t size\_of\_element;

} stack\_on\_array\_t;

Здесь сами данные хранятся в динамически расширяемом векторе data, указатель head указывает на последний элемент в данных, count и max\_count – текущее количество элементов и максимальное соответственно, size\_of\_element – размер элементов. У данной структуры есть следующие особенности:

* данные в памяти хранятся последовательно;
* выделение памяти под новые элементы происходит блоками (при достижении максимума размер выделенной памяти умножается на 2, а элементы копируются из старой области в новую);
* при удалении сдвигается только указатель на последний элемент;

Вспомогательной структурой является вектор, который реализует хранение данных. Его описание на языке C выглядит следующим образом:

typedef struct

{

void \*front;

size\_t size\_of\_element;

size\_t size;

size\_t alloc\_size;

} vector\_t;

1. **Описание алгоритма**

***Оценка по времени***

Для обработки стека (под обработкой имеется в виду определение правильности расстановки скобок в выражении) были использованы стеки, реализованные на основе списка и на основе вектора. Исходя из особенностей каждой структуры, мы можем сделать вывод о том, что определённо эффективнее

***Оценка по памяти***

***Итог***

***Проверка выводов экспериментально***

**Тестирование**

1. Позитивные тесты

|  |  |  |
| --- | --- | --- |
| Входные данные | Описание теста | Результат |
| 0 0 4 10  6 0 0 0  0 0 3 0  0 4 0 8  0 8 0 0 | Проверка умножения при стандартном хранении | 48 0 0 0 |
| data:  4 10 6 3 4 8  columns:  2 3 0 2 1 3  pointers:  0 2 3 4 6  data:  8  columns:  2 | Проверка умножения при сжатом хранении | data:  48  columns:  1 |
| 5 4 2 2 6  4 3 3 9 3  8 10 7 9 9  5 6 9 5 2  7 4 2 7 10  5 0 0 0 0 | Проверка умножения при стандартном хранении | 25 20 10 10 30 |
| data:  5 4 2 2 6 4 3 3 9 3 8 10 7 9 9 5 6 9 5 2 7 4 2 7 10  columns:  0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4  pointers:  0 5 10 15 20 25  data:  5  columns:  1 | Проверка умножения при сжатом хранении | data:  25 20 10 10 30  columns:  1 2 3 4 5 |

1. **Выводы по проделанной работе**

В ходе работы был проведён сравнительный анализ подходов к хранению и обработке разреженных матриц и векторов (на примере операции умножения), в результате которого установилось, что при маленьком проценте заполнения объектов (при большом количестве нулевых элементов) выгоднее по памяти и по времени использовать сжатый способ хранения. При достаточно заполненных объектах при сжатом способе хранения возникают лишние затраты по времени и памяти, поэтому разумно использовать стандартный способ хранения.

1. **Ответы на вопросы**

1. Что такое разреженная матрица, какие схемы хранения таких матриц Вы

знаете?

Разреженная матрица – матрица, содержащая преимущественно нули. Хранить можно стандартно (все элементы) и с помощью следующей схемы:

- вектор A содержит значения ненулевых элементов;

- вектор JA содержит номера столбцов для элементов вектора A;

- связный список IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A.).

1. Каким образом и сколько памяти выделяется под хранение разреженной и обычной матрицы?

См. пункт “Оценка по памяти” в описании алгоритмов.

1. Каков принцип обработки разреженной матрицы?

См. пункт “Сжатый способ хранения” в описании алгоритмов.

1. В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?

См. пункт “Итог” в описании алгоритмов.