Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: Информатика и системы управления
КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии
ДИСЦИПЛИНА: Типы и структуры данных
ТЕМА: Работа со стеком
ВАРИАНТ: 3

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Студент: _____ *Княжев А. В.*
(подпись, дата)

Преподаватель: _____ *Силантьева А. В.*
(подпись, дата)

2021 г.

Содержание

1 Условие задачи

Элементами стека являются слова. Распечатайте слова в обратном порядке в перевернутом виде.

2 Техническое задание

Разработать программу работы со стеком, реализующую операции добавления и удаления элементов из стека и отображения текущего состояния стека. Реализовать стек:

* массивом;
* списком.

Все стандартные операции со стеком должны быть оформлены отдельными подпрограммами. В случае реализации стека в виде списка при отображении текущего состояния стека предусмотреть возможность просмотра адресов элементов стека и создания дополнительного собственного списка свободных областей (адресов освобождаемой памяти при удалении элемента, который можно реализовать как списком, так и массивом) с выводом его на экран. Список свободных областей необходим для того, чтобы проследить, каким образом происходит выделение памяти менеджером памяти при запросах на нее и убедиться в возникновении или отсутствии фрагментации памяти.

Сравнить эффективность (по памяти и по времени выполнения) обработки стека списком и массивом.

Программа должна выводить меню с возможностью выбирать варианты. При вводе нуля на запрос варианта меню, происходит выход из программы.

2.1 Общие входные данные

* номер выбранного пункта меню.

2.2 Входные и выходные данные пунктов меню

2.2.1 Добавление элемента в стек-массив

Входные данные

* строка (длиной до 256 символов), добавляемая в стек.

Выходные данные

* сообщение об успешности добавления.

2.2.2 Добавление элемента в стек-список

Входные данные

* строка (длиной до 256 символов), добавляемая в стек.

Выходные данные

* сообщение об успешности добавления.

2.2.3 Удаление элемента из стека-массива

Выходные данные

* значение удаленной строки.

2.2.4 Удаление элемента из стека-списка

Выходные данные

* значение удаленной строки.

2.2.5 Вывод перевернутых строк в обратном порядке из стека-массива

Выходные данные

* список перевернутых строк, в обратном порядке относительно добавления, которые были в стеке.

2.2.6 Вывод перевернутых строк в обратном порядке из стека-списка

Выходные данные

* список перевернутых строк, в обратном порядке относительно добавления, которые были в стеке.

2.2.7 Вывод состояния стека-массива

Выходные данные

* список строк, хранящихся в стеке (сверху те, что были добавлены последними).

2.2.8 Вывод состояния стека-списка

Выходные данные

* список строк, хранящихся в стеке (сверху те, что были добавлены последними);
* для каждого элемента адрес элемента стека-списка.

2.2.9 Вывод списка свободных областей для стека-списка

Выходные данные

* список адресов областей, освобожденных в результате удаления элементов из стека-списка.

2.2.10 Сравнение скорости работы стеков двух типов

Входные данные

* количество элементов.

Выходные данные

* скорости работы добавления и удаления введенного количества элементов для каждого вида стека и их сравнения.

2.2.11 Сравнение объема занимаемой памяти стеков двух типов

Входные данные

* количество элементов.

Выходные данные

* объемы занимаемой памяти стеков с указанным количеством элементов и их сравнение.

2.3 Действие программы

Программа осуществляет работу со стеками в формате массива и списка.

2.4 Обращение к программе

Программа может быть запущена из командных оболочек `sh/bash/zsh/fish`, а также от IDE, способных работать с языком Си. Программа не принимает никаких аргументов.

2.5 Аварийные ситуации

2.5.1 Общие аварийные ситуации

1. невалидный номер пункта меню (строка или пустая строка);
2. неверный номер пункта меню (такого пунта меню не существует).

2.5.2 Пункт меню №1

1. введена пустая строка;
2. введена строка длиной больше 256 символов.

2.5.3 Пункт меню №2

1. введена пустая строка;
2. введена строка длиной больше 256 символов.

2.5.4 Пункт меню №3

1. стек пуст.

2.5.5 Пункт меню №4

1. стек пуст.

2.5.6 Пункт меню №5

1. стек пуст.

2.5.7 Пункт меню №6

1. стек пуст.

2.5.8 Пункт меню №7

1. стек пуст.

2.5.9 Пункт меню №8

1. стек пуст.

2.5.10 Пункт меню №9

2.5.11 Пункт меню №10

1. слишком большое количество элементов;
2. слишком маленькое количество элементов;
3. невалидное количество элементов.

2.5.12 Пункт меню №11

1. слишком большое количество элементов;
2. слишком маленькое количество элементов;
3. невалидное количество элементов.

3 Структуры данных

В данной работе используется статичесике массивы, записи и односвязные списки.

3.1 Модуль для работы с ошибками

3.1.1 Типы данных

```
/*
 * text - текст ошибки;
 * code - код ошибки;
 * func - функция, в которой случилась ошибка.
 */
typedef struct
{
    const char *text;
    int code;
    const char *func;
} error_t;
```

3.1.2 Функции

```
/*
 * Создание новой ошибки.
 * text - текст ошибки;
 * code - код ошибки;
 * func - функция, в которой случилась ошибка.
 */
error_t new_error(const char *text, int code, const char *func);
```

```
/*
 * Создание ошибки-маркера успеха.
 * func - функция, в которой был создан "успех".
 */
error_t new_success(const char *func);
```

```c
/*
 * Проверка, отображает ли ошибка ошибочную ситуацию.
 * err - исходная ошибка.
 */
bool is_failure(error_t err);
```

3.2 Модуль для работы со строками

3.2.1 Основные константы

```c
/*
 * Тип строки длиной 256 символов + терминальный нуль.
 */
typedef char mystring_t[MYSTRING_SIZE];
```

3.2.2 Типы данных

```c
/*
 * Тип строки длиной 256 символов + терминальный нуль.
 */
typedef char mystring_t[MYSTRING_SIZE];
```

3.2.3 Функции

```c
/*
 * Чтение строки из файла.
 * f - файл;
 * str - строка, в которую считываем.
 */
error_t f_read_line(FILE *f, mystring_t str);
```

```c
/*
 * Чтение целого числа из стандартного потока ввода.
 * n - считываемое число.
 */
error_t read_int(int *n);
```

```
/*
 * Переворачивает строку.
 * str - исходная строка.
 */
error_t reverse(char *str);
```

3.3 Модуль для работы со списком указателей

3.3.1 Типы данных

```
/*
 * Список указателей.
 * content - указатель (содержимое элемента);
 * next - указатель на следующий элемент.
 */
typedef struct ptr_list_t ptr_list_t;
struct ptr_list_t
{
    void *content;
    ptr_list_t *next;
};
```

3.3.2 Функции

```
/*
 * Добавление элемента в список.
 * head - голова списка;
 * ptr - добавляемый указатель.
 */
error_t pl_add(ptr_list_t **head, void *ptr);
```

```
/*
 * Удаление указателя.
 * head - голова списка;
 * ptr - удаляемый указатель.
 */
error_t pl_delete(ptr_list_t **head, void *ptr);
```

```c
/*
 * Вывод элементов списка.
 * head - голова списка.
 */
error_t pl_print(ptr_list_t **head);
```

```c
/*
 * Освобождение списка.
 * head - голова списка.
 */
void pl_free(ptr_list_t **head);
```

3.4 Модуль «ядра» программы

3.4.1 Функции

```c
/*
 * Запуск основного диалога программы.
 */
error_t eng_work();
```

3.5 Модуль для тестирования скорости работы

3.5.1 Основные константы

```c
/*
 * Количество запусков тестов производительности.
 */
#define BENCH_NUM_OF_RUNS 10000
```

3.5.2 Функции

```c
/*
 * Получение скорости n операций "push" для стека-массива.
 * n - количество элементов;
 * timer - среднее время.
 */
error_t bench_stack_arr_push(size_t n, long long *timer);
```

```c
/*
 * Получение скорости n операций "pop" для стека-массива.
 * n - количество элементов;
 * timer - среднее время.
 */
error_t bench_stack_arr_pop(size_t n, long long *timer);
```

```c
/*
 * Получение скорости n операций "push" для стека-списка.
 * n - количество элементов;
 * timer - среднее время.
 */
error_t bench_stack_list_push(size_t n, long long *timer);
```

```c
/*
 * Получение скорости n операций "pop" для стека-списка.
 * n - количество элементов;
 * timer - среднее время.
 */
error_t bench_stack_list_pop(size_t n, long long *timer);
```

3.6 Модуль для работы со стеком-массивом

3.6.1 Основные константы

```c
/*
 * Максимальное количество элементов стека-массива.
 */
#define SA_STACK_SIZE 5000
```

3.6.2 Типы данных

```
/*
 * Стек-массив.
 * content - содержимое стека;
 * ptr - указатель на место, в которое вставлять очередной
 *       элемент стека.
 */
typedef struct stack_arr_t
{
    mystring_t content[SA_STACK_SIZE];
    mystring_t *ptr;
} stack_array_t;
```

3.6.3 Функции

```
/*
 * Создание пустого стека.
 */
stack_array_t sa_create();
```

```
/*
 * Добавление элемента в стек.
 * sa - исходный стек;
 * str - добавляемая строка.
 */
error_t sa_push(stack_array_t *sa, mystring_t str);
```

```
/*
 * Удаление элемента из стека.
 * sa - исходный стек;
 * str - значение удаленной строки.
 */
error_t sa_pop(stack_array_t *sa, mystring_t str);
```

```
/*
 * Проверка стека на пустоту.
 * sa - исходный стек.
 */
bool sa_empty(stack_array_t *sa);
```

3.7 Модуль для работы со стеком-списком

3.7.1 Типы данных

```
/*
 * Стек-список.
 * content - содержимое элемента стека (строка);
 * next - указатель на следующий элемент стека.
 */
typedef struct stack_list_t stack_list_t;
struct stack_list_t
{
    mystring_t content;
    stack_list_t *next;
};
```

3.7.2 Функции

```
/*
 * Добавление элемента в стек.
 * sl - исходный стек;
 * str - добавляемая строка.
 */
error_t sl_push(stack_list_t **sl, mystring_t str);
```

```
/*
 * Удаление элемента из стека.
 * sl - исходный стек;
 * str - значение удаленной строки.
 */
error_t sl_pop(stack_list_t **sl, mystring_t str);
```

```c
/*
 * Освобождение стека-списка.
 * sl - исходный стек.
 */
error_t sl_free(stack_list_t **sl);
```

```c
/*
 * Проверка стека на пустоту.
 * sl - исходный стек.
 */
bool sl_empty(stack_list_t **sl);
```

4 Описания алгоритмов

1. вывод главного меню;
2. получение у пользователя номера пункта меню:

4.1 Пункт меню №1

1. получить у пользователя строку;
2. добавить ее в стек-массив.

4.2 Пункт меню №2

1. получить у пользователя строку;
2. добавить ее в стек-список.

4.3 Пункт меню №3

1. удалить элемент из стека-массива;
2. вывести удаленный элемент.

4.4 Пункт меню №4

1. удалить элемент из стека-списка;
2. добавить адрес удаленного элемента в список свободных областей;
3. вывести удаленный элемент.

4.5 Пункт меню №5

1. пока стек-массив не пуст:
   (a) вытащить элемент из стека-массива;
   (b) перевернуть этот элемент (как строку);
   (c) вывести перевернутую строку на экран.

4.6 Пункт меню №6

1. пока стек-список не пуст:
   (a) вытащить элемент из стека-списка;
   (b) перевернуть этот элемент (как строку);
   (c) вывести перевернутую строку на экран.

4.7 Пункт меню №7

1. вывести список элементов стека-массива.

4.8 Пункт меню №8

1. вывести список элементов стека-списка с адресами.

4.9 Пункт меню №9

1. вывести список свободных областей (адресов в памяти, которые были получены после удаления очередных элементов из стека-списка и пока не заняты).

4.10 Пункт меню №10

1. получить у пользователя количество элементов;
2. для каждого вида стека посчитать среднюю скорость добавления и удаления введенного количества элементов и вывести на экран;
3. вывести процентное превосходство скорости работы стека-массива над стеком-списоком.

4.11 Пункт меню №11

1. получить у пользователя количество элементов;
2. для каждого вида стека посчитать количество памяти, затрачиваемое для хранения введенного количества элементов;
3. вывести процентное превосходство объема занимаемой памяти стека-списка над списком-массивом.

5 Тестирование

Для проверок корректности работы программы было проведено функциональное тестирование. Таблица с тестовыми данными для «позитивных» и «негативных» случаев приведена ниже.

Так как входных/выходных данных может быть очень много, то в соответствующих полях могут быть указаны наиболее значимые части.

5.1 «Негативные» тесты

5.1.1 Ввод невалидного пункта меню

Входные данные: abc

Стек-массив: —

Стек-список: —

Результат: 104: Строка не является корректным числом (to_integer)

5.1.2 Ввод некорректного пункта меню

Входные данные: 12

Стек-массив: —

Стек-список: —

Результат: 161: введен неверный пункт меню (main_menu_dialog)

5.1.3 Ввод слишком длинной строки

Входные данные: 1 → *<строка из 300 символов>* или 2 → *<строка из 300 символов>*

Стек-массив: —

Стек-список: —

Результат: 103: Считанная строка слишком длинная (f_read_line)

5.1.4 Удаление элемента из пустого стека

Входные данные: 3 или 4

Стек-массив: —

Стек-список: —

Результат: 101: стек пуст (sa_pop) или 122: стек пуст (sl_pop)

5.1.5 Вывод перевернутых слов из пустого стека

Входные данные: 5 или 6

Стек-массив: —

Стек-список: —

Результат: 140: стек-массив пуст (sa_print_reversed) или 140: стек-список пуст (sl_print_reversed)

5.1.6 Вывод состояния пустого стека

Входные данные: 7 или 8

Стек-массив: —

Стек-список: —

Результат: 140: стек-массив пуст (sa_print_state) или 140: стек-список пуст (sl_print_state)

5.1.7 Ввод невалидного количества элементов

Входные данные: 10 → abc или 11 → abc

Стек-массив: —

Стек-список: —

Результат: 104: Строка не является корректным числом (to_integer)

5.1.8 Ввод слишком большого количества элементов

Входные данные: 10 → 100000 или 11 → 100000

Стек-массив: —

Стек-список: —

Результат: 100: слишком большой размер (eng_work)

5.1.9 Ввод слишком маленького количества элементов

Входные данные: 10 → -2 или 11 → -2

Стек-массив: —

Стек-список: —

Результат: 161: неверное количество элементов стека (eng_work)

5.1.10 Переполнение стека-массива

Входные данные: 1 → one

Стек-массив: стек с максимальным количеством элементов

Стек-список: —

Результат: 100: переполнение стека (sa_push)

5.2 «Позитивные» тесты

5.2.1 Добавление элемента в стек-массив

Входные данные: 1 → one

Стек-массив: —

Стек-список: —

Результат: Строка успешно добавлена.

5.2.2 Добавление элемента в стек-список

Входные данные: 2 → two

Стек-массив: —

Стек-список: —

Результат: Строка успешно добавлена.

5.2.3 Удаление элемента из стека-массива

Входные данные: 3

Стек-массив: [hello, one]

Стек-список: —

Результат: Удаленная строка: "hello"

5.2.4 Удаление элемента из стека-списка

Входные данные: 4

Стек-массив: —

Стек-список: [four, three, two]

Результат: Удаленная строка: "four"

5.2.5 Вывод состояния стека-массива

Входные данные: 7

Стек-массив: [hello, world, my, name, is, alex]

Стек-список: —

Результат:

```
Состояние стека-массива:
"hello"
"world"
"my"
"name"
"is"
"alex"
```

5.2.6 Вывод перевернутых строк в обратном порядке из стека-массива

Входные данные: 5

Стек-массив: [hello, world, my, name, is, alex]

Стек-список: —

Результат:

```
"olleh"
"dlrow"
"ym"
"eman"
"si"
"xela"
```

5.2.7 Вывод состояния стека-списка

Входные данные: 8

Стек-массив: —

Стек-список: [five, four, three, two, one]

Результат:

```
Состояние стека-списка:
0x14b004190 "five"
0x14b104080 "four"
0x14b004080 "three"
0x149f04190 "two"
0x149f04080 "one"
```

5.2.8 Вывод перевернутых строк в обратном порядке из стека-списка

Входные данные: 6

Стек-массив: —

Стек-список: [five, four, three, two, one]

Результат:

```
"evif"
"ruof"
"eerht"
"owt"
"eno"
```

5.2.9 Вывод списка свободных областей

Входные данные: 9

Стек-массив: —

Стек-список: —

Результат:

```
Список свободных областей для стека-списка:
0x14b1044c0
0x14b1043b0
0x14b1042a0
0x14b104080
0x14b104190
0x149f04080
0x149f04190
0x14b004080
0x14b004190
```

5.2.10 Сравнение стеков по скорости

Входные данные: $10 \rightarrow 1000$

Стек-массив: —

Стек-список: —

Результат:

```
Для 10000 запусков и 1000 элементов.
Стек-массив
  PUSH: 16 us
  POP: 15 us
Стек-список
  PUSH: 37 us
  POP: 37 us

Преимущество стека-массива над стеком-списком
  PUSH: 131 %
  POP: 147 %
```

5.2.11 Сравнение стеков по памяти

Входные данные: $11 \rightarrow 1000$

Стек-массив: —

Стек-список: —

Результат:

```
Для 1000 элементов.
Стек-массив
  Объем занимаемой памяти: 1285008 B

Стек-список
  Объем занимаемой памяти: 272000 B

Преимущество стека-списка над стеком-массивом
  Объем памяти: 372 %
```

6 Оценка эффективности

6.1 Объем занимаемой памяти

Размер	Список, Б	Массив, Б	Преимущество списка, %
50	13600	1285008	9349
100	27200	1285008	4624
200	54400	1285008	2262
400	108800	1285008	1081
800	217600	1285008	491
1600	435200	1285008	195
3200	870400	1285008	48
5000	1360000	1285008	-6

6.2 Скорость операций

6.2.1 Push

Размер	Массив, мкс	Список, мкс	Преимущество массива, %
50	2	2	0
100	3	5	67
200	7	8	14
400	14	15	7
800	24	29	21
1600	40	58	45
3200	66	123	86
5000	94	207	120
10000	163	478	193

6.2.2 Pop

Размер	Массив, мкс	Список, мкс	Преимущество массива, %
50	1	2	100
100	2	4	100
200	4	8	100
400	8	15	88
800	13	30	131
1600	24	60	150
3200	48	122	154
5000	75	209	179
10000	154	401	160

7 Контрольные вопросы

7.1 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, Last In – First Out (LIFO).

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

При хранении стека, как списка, память выделяется в куче, при этом памяти выделяется именно столько, сколько сейчас элементов в стеке. С каждым элементом стека, хранится указатель на следующий элемент, что увеличивает объем памяти.

При хранении стека, как статического массива, память выделяется на стеке, ее выделяется какое-то изначально заданное количество (под 5000, 10000 элементов, и т. д.). Кроме самих элементов, в этом случае, необходимо хранить указатель на текущий элемент стека.

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

При хранении стека, как списка происходит освобождение верхнего элемента и сдвиг указателя на следующий.

При хранении стека, как массива, происходит сдвиг указателя к следующему элементу.

7.4 Что происходит с элементами стека при его просмотре?

Элементы удаляются.

7.5 Каким образом эффективнее реализовывать стек? От чего это зависит?

Если наиболее важна скорость обработки, и не так важен объем занимаемой памяти, то целесообразнее использовать реализацию стека в виде массива.

Иначе, целесообразнее использовать реализацию стека в виде списка, так как в этом случае занимаемый объем памяти соответствует количеству элементов в стеке, а также

элементы стека не обязаны располагаться в памяти друг за другом, то есть увеличивается «надежность» выделения памяти: проще найти несколько разделенных маленьких фрагментов памяти, чем искать один большой цельный кусок.

8 Вывод

По памяти на небольших размерах эффективнее использовать стек-список, чем стек-статический массив, так как размер списка зависит только от количества его текущих элементов. Однако, на больших размерах, близких к максимальному размеру стека-массива, массив начинает быть более эффективным, так как массив хранит в себе только сами элементы, а список же содержит еще и дополнительную информацию, такую как указатель на следующий элемент. Преимущество по памяти у стека-списка снижается с 9349 % на 50 элементах, до -6 % на 5000 элементах.

По времени эффективнее использовать стек-массив, так как операция «push» для списка несет в себе выделение памяти, а «pop» — освобождение. Эти операции являются довольно долгими, поэтому такие операции менее эффективны, по сравнению с простым сдвигом указателя в стеке-массиве. Преимущество по скорости у стека-массива варьируется от 0 до 193 % для операции «push» и от 88 до 179 % для операции «pop».

В ходе тестирования программы было обнаружено, что иногда происходит фрагментация данных. Это плохо, так как преимуществ у фрагментации нет. Преимуществами дефрагментации данных являются более высокая скорость и простота доступа к данным. Так как операционная система предоставляет нам необходимый интерфейс, то работа с фрагментарованными и дефрагментированными данными для пользователя одинакова.

Список литературы

[1] Методические рекомендации по лабораторной работе №4 (*http://wwwcdl.bmstu.ru/*)