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 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

РАБОТА СО СТЕКОМ

Студент Жаринов М. А.

Вариант 5

Группа **ИУ7-32Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Жаринов М. А.**

Преподаватель _____ **Барышникова М. Ю.**

2024

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 стека и создать свой список или массив свободных областей (адресов освобождаемых элементов) с выводом его на экран.

Ввести арифметическое выражение типа: число|знак| … |число|знак| число. Вычислить значение выражения.

Провести сравнение разных реализаций стека по памяти и скорости работы

Техническое задание

Исходные данные:

Номер пункта меню: Целое число от 0 до 8.

Выражение-пример: Строка вида число|знак| … |число|знак| число

Выходные данные:

Значение выражения, результаты сравнения способов реализации.

Реализуемые задачи:

Пункт меню 0 – Выход из программы

Пункт меню 1 – Инициализация стека на массиве

Пункт меню 2 – Инициализация стека на списке

Пункт меню 3 – Запись элемента на стек

Пункт меню 4 – Снятие элемента со стека

Пункт меню 5 – Печать стека

Пункт меню 6 – Печать массива освобожденных адресов (только для стека на списке)

Пункт меню 7 – Расчет значения выражения

Пункт меню 8 – Проведение сравнительного анализа различных методов реализации стека

Способ обращения к программе:

Строка запуска программы ./app.exe

После запуска с помощью меню нужно инициализировать стек, ввести выражение или запустить анализ методов реализации.

Аварийные ситуации:

1. EOF в вводе. Код ошибки 1(ERR_EOF)

Все остальные ошибочные ситуации (некорректные элементы к записи, не инициализированный стек, некорректное выражение, некорректный пункт меню, деление на 0 в выражении) не являются аварийными и вызывают повторное приглашение к вводу или возвращение в меню.

Внутренние структуры данных

Элементы стека хранится в структуре `token_t`.

```
typedef struct
{
    bool is_operator;
    union
    {
        double operand;
        char operator;
    } value;
} token_t;
```

Поле `is_operator` – Определяет тип элемент

Поле `value.operand`– вещественное число

Поле `value.operator`– символ оператора

Вершины списка хранятся в структуре `list_node_t`.

```
struct list_node_t
{
    token_t value;
    list_node_t *next_node;
};
```

Поле `value` –значение элемента

Поле `next_node` – указатель на следующий элемент

Стек в виде массива хранятся в структуре `array_stack_t`.

```
typedef struct
{
    token_t tokens[STACK_DEPTH];
    size_t depth;
} array_stack_t;
```

Поле `tokens` –массив элементов

Поле `depth` –глубина стека

Стек в виде списка хранятся в структуре `list_stack_t`.

```
typedef struct
{
    list_node_t *head;
```

```
    size_t depth;
} list_stack_t;
```

Поле `head` –указатель на вершину списка

Поле `depth` –глубина стека

Массив освобожденных адресов памяти хранятся в структуре `freed_nodes_array_t`.

```
typedef struct
{
    list_node_t *freed_nodes[STACK_DEPTH];
    size_t len;
} freed_nodes_array_t;
```

Поле `freed_nodes` –массив указателей на элементы списка

Поле `len` –длина массива

Для удобства работы используется абстрактный тип для использования стека независимо от его реализации `debug_stack_t`.

```
typedef struct
{
    void *stack;
    bool (*push)(void *stack, token_t value);
    bool (*pop_mem)(void *stack, token_t *value, freed_nodes_array_t
*freed_memory);
    bool (*print)(void *stack);
    freed_nodes_array_t *freed_memory;
    void (*print_freed)(freed_nodes_array_t *freed_memory);
    void (*free)(void *stack, freed_nodes_array_t *freed_memory);
} debug_stack_t;
```

Поле `stack` – указатель на стек

Поле `push` – указатель на функцию записи элемента на стек

Поле `pop_mem` – указатель на функцию снятия элемента со стека с сохранением освобожденного адреса

Поле `print` – указатель на функцию печати стека

Поле `freed_memory` – указатель на массив освобожденных адресов

Поле `print_freed` – указатель на функцию печати массива освобожденных адресов

Поле `free` – указатель на функцию освобождения стека

Также для корректного замера скорости работа создана упрощенная структура стека без лишних функций и сохранения адресов освобожденной памяти `stack_t`.

```
typedef struct
{
    void *stack;
    bool (*push)(void *stack, token_t value);
    bool (*pop)(void *stack, token_t *value);
    void (*free)(void *stack);
} stack_t;
```

Поле `stack` – указатель на стек

Поле `push` – указатель на функцию записи элемента на стек

Поле `pop` – указатель на функцию снятия элемента со стека

Поле `free` – указатель на функцию освобождения стека

Описание алгоритма

Программа запрашивает номер элемента в меню, считывает его и выполняет соответствующее действие:

Пункт меню 0 – Выход из программы

Пункт меню 1 – Инициализация стека на массиве

Пункт меню 2 – Инициализация стека на списке

Пункт меню 3 – Запись элемента на стек

1. Программа запрашивает и считывает токен, определяет его тип
2. Программа записывает элемент на стек

Пункт меню 4 – Снятие элемента со стека

Пункт меню 5 – Печать стека

Пункт меню 6 – Печать массива освобожденных адресов (только для стека на списке)

Пункт меню 7 – Расчет значения выражения

1. Программа запрашивает и считывает выражение в строку
2. Программа создает 2 стека, реализация выбирается случайно
3. Программа считывает выражение в 1 стек
4. Программа переносит выражение из 1 стека во 2 для исправления порядка выражения
5. Программа переносит элементы из 2 стека в 1, считая при этом операции с большим приоритетом
6. Программа переносит выражение из 1 стека во 2 для исправления порядка выражения
7. Программа переносит элементы из 2 стека в 1 и считает оставшиеся операции
8. Программа проверяет и печатает ответ, хранящийся в единственном элементе 1 стека

Пункт меню 8 – Проведение сравнительного анализа различных методов реализации стека

1. Программа в цикле задает нужные размеры выражения

2. Программа создает случайное выражение.

3. Программа находит среднее время вычисления выражения с двумя методами реализации стека, выигрыш при использовании стека на массиве, используемый размер стека на списке, выигрыш памяти при использовании стека на списке и печатает результаты.

Основные функции

Функция для обработки выбранного пункта меню. Принимает номер выбранного пункта меню и указатель на стек. Возвращает код ошибки

```
int process_menu(menu_item_t menu_item, debug_stack_t *debug_stack);
```

Функция записи элемента на стек-массив. Принимает указатель на стек, элемент для записи. Возвращает флаг успешности выполнения

```
bool push_array(void *array_stack, token_t value);
```

Функция снятия элемента со стека-массива. Принимает указатель на стек, указатель на элемент для чтения. Возвращает флаг успешности выполнения

```
bool pop_array(void *array_stack, token_t *value);
```

Функция печати стека-массива. Принимает указатель на стек

```
bool print_array(void *array_stack);
```

Функция освобождения стека-массива. Принимает указатель на стек

```
void free_array(void *array_stack);
```

Функция инициализации абстрактного стека так, чтобы он был стеком-массивом. Принимает указатель на абстрактный стек

```
void init_array_debug(debug_stack_t *debug_stack);
```

Функция записи элемента на стек-список. Принимает указатель на стек, элемент для записи. Возвращает флаг успешности выполнения

```
bool push_list(void *list_stack, token_t value);
```

Функция снятия элемента со стека-списка. Принимает указатель на стек, указатель на элемент для чтения. Возвращает флаг успешности выполнения

```
bool pop_list(void *list_stack, token_t *value);
```

Функция печати стека-списка. Принимает указатель на стек

```
bool print_list(void *list_stack);
```

Функция освобождения стека-списка. Принимает указатель на стек

```
void free_list(void *list_stack);
```

Функция инициализации абстрактного стека так, чтобы он был стеком-списком. Принимает указатель на абстрактный стек

```
void init_list_debug(debug_stack_t *debug_stack);
```

Функция вычисления значения выражения. Принимает указатели на 2 стека, в первом хранится выражение в порядке считывания. Ответ находится в единственном элементе 1 стека

```
void solve_problem(stack_t *in_stack, stack_t *out_stack);
```

Функция анализа разных реализаций стека

```
void run_analysis(void);
```

Набор тестов

Обработка пунктов меню

Описание теста	Вход	Результат
Отрицательное число	-1	Для выбора пункта меню введите целое число от 0 до 8
Число больше 8	9	Для выбора пункта меню введите целое число от 0 до 8
Пустой ввод		Для выбора пункта меню введите целое число от 0 до 8
Максимальный пункт меню	8	[сведения о скорости вычислений]
Минимальный пункт меню	0	[выход из программы]
Символ вместо числа	a	Для выбора пункта меню введите целое число от 0 до 8
Выбор пунктов меню, которые требуют инициализированный стек, при неинициализированном стеке	3	Нельзя производить данное действие над неинициализированным стеком!
Снятие элемента со стека при пустом стеке	4	Стек пуст!
Запись на стек при полном стеке	3 1.2	Стек переполнен!

Запись элемента на стек

Описание теста	Ввод	Вывод
Ввод целого числа	3	В стек успешно добавлено число
Ввод вещественного числа	.5e5	В стек успешно добавлено число

Ввод оператора	+	В стек успешно добавлен оператор
Ввод некорректного оператора	^	Введено некорректное значение!
Ввод оператора с посторонним символом	-)	Введено некорректное значение!
Ввод числа с посторонним символом	3e	Введено некорректное значение!
Ввод числа с 2 знаками	++3	Введено некорректное значение!
Пустой ввод		Введено пустое значение!

Расчет значения выражения

Описание теста	Ввод	Вывод
Ввод одного числа	3	3.000000
Ввод простого выражения	1+1*2/4-0.5e1	1.000000
Ввод выражения максимальной длины	1-1+1-1…-1 (50001) элемент	0.000000
Пустой ввод		Введено некорректное выражение!
Ввод выражения больше максимальной длины	1-1+1-1…-1+1 (50001) элемент	Введено некорректное выражение!
Ввод выражения с делением на 0	1+3/0.0e23	Деление на 0 в выражении!
Ввод выражения с неправильным содержанием	1+4a-4/4	Введено некорректное выражение!

Результаты сравнения

Для анализа я измерил средние скорости вычисления значения выражения при длине выражения 5, 51, 501, 5001 и 50001 элементов и получил следующие результаты (в наносекундах)

Длина выражения	Время расчета с использованием стека на списке	Время расчета с использованием стека на массиве	Выигрыш времени от стека на массиве	Размер стека на списке, байт
5	371	157	57.68%	136
51	2816	1036	63.21%	1240
501	29500	10908	63.02%	12040
5001	303481	122050	59.78%	120040
50001	3777733	1248051	66.96%	1200040

Размер стека на массиве, байт: 800024

Таким образом, при использовании стека, реализованного на массиве, время обработки выражения уменьшается на 55-70 процентов, и чем больше выражение, тем больше выигрыш.

При использовании стека не на максимальную глубину, стек на списке занимает гораздо меньше памяти, так как он выделяет память только под реально нужное количество элементов. Однако при использовании стека на максимальную глубину стек на списке занимает на 50% больше памяти, так как он хранит не только сами элементы, но и указатели на каждый элемент.

Ответы на контрольные вопросы

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 (LIFO).

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

При реализации стека через статический массив всегда выделяется (макс. кол-во элементов)*(размер элемента) = 800КБ, а при реализации через список динамически выделяется от 40 байт до 1.2МБ (реальное кол-во элементов)*(размер элемента + размер указателя)

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

При реализации стека через статический массив при удалении память не освобождается, а при реализации через список освобождается память из под вершины.

4. Что происходит с элементами стека при его просмотре?

При «честном» просмотре элемента стека происходит сначала снятие элемента со стека, далее его обработка, и затем запись элемента обратно на стек

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

Способ, которым реализовывать стек эффективнее, зависит от того, какие данные планируется хранить и что важнее – быстродействие или память. Если важнее быстродействие или если элементы стека имеет размер сравнимый или меньший чем размер указателя и известен максимальный размер стека, то эффективнее использовать статический массив.

Если важнее меньшее использование памяти или максимальная глубина стека не ограничена, то имеет смысл использовать список

Вывод

При решении задач, которые требуют только возможности записи элемента в конец и чтения элемента с конца, таких как вычисление значения выражения, целесообразно вместо обычной структуры данных, такой как массив или список, использовать абстрактный тип данных – стек, так как он позволяет менять его реализацию без изменения основного кода программы.

Способ, которым реализовывать стек эффективнее, зависит от того, какие данные планируется хранить и что важнее – быстродействие или память. Если важнее быстродействие или если элементы стека имеет размер сравнимый или меньший чем размер указателя и известен максимальный размер стека, то эффективнее использовать статический массив.

Если важнее меньшее использование памяти или максимальная глубина стека не ограничена, то имеет смысл использовать список

При использовании стека, реализованного на массиве, время обработки выражения уменьшается на 55-70 процентов, и чем больше выражение, тем больше выигрыш.

При использовании стека не на максимальную глубину, стек на списке занимает гораздо меньше памяти, так как он выделяет память только под реально нужное количество элементов. Однако при использовании стека на максимальную глубину стек на списке занимает на 50% больше памяти, так как он хранит не только сами элементы, но и указатели на каждый элемент.

Также, при использовании стека на списке, память обычно не фрагментируется – если в цикле производить запись и снятие элемента со стека, то он почти всегда будет записываться по одному и тому же адресу.