Условие задачи

Разработать программу для работы со стеком, поддерживающую базовые операции, такие как добавление, удаление и отображение элементов. Стек должен быть реализован двумя способами: на основе статического массива и на основе динамического односвязного списка. Все операции со стеком должны быть реализованы как подпрограммы. При реализации стека как односвязного списка необходимо предусмотреть возможность вывести массив свободных областей памяти, который должен формироваться в процессе работы с таким стеком. Предусмотреть операцию ввода арифметического выражения типа: число|знак|...число|знак| число, а также вычисления значения этого выражения. Приоритетность операций не учитывается.

Описание ТЗ

Исходные данные

Исходными данными программы являются:

1. Строка, содержащее единственное целое положительное число, являющееся пунктом меню.
2. Строка, содержащая целые числа и операторы (под операторами имеются в виду «+, -, *, /», т.е. «плюс», «минус», «умножить», «разделить») в формате «число|знак|...число|знак|», например, «3+6-7*2».
3. Строка, содержащая единственное целое число.

На исходные данные накладываются следующие ограничения:

1. Числа, запрашиваемые программой (при вводе пункта меню или добавления элемента в стек), должны быть целыми. В случае с вводом пункта меню число должно быть от 0 до 11.
2. В случае реализации стека на основе статического массива размер стека ограничен константой MAX_ARR_STACK_SIZE, в текущей реализации это значение равно 100.
3. Строка, в которую вводится пункт меню, число для добавления в стек, а также арифметическое выражение, имеет ограничение по длине – 256 символов.

Результат

Результатом выполнения программы являются:

1. Вычисленное значение арифметического выражения.
2. Извлеченный элемент из стека.

3. Текущее состояние стека.
4. Таблица сравнения производительности вычисления значения вводимого выражения на основе двух способов реализации стека.
5. Таблица сравнения производительности операций push и pop на основе двух способов реализации стека.
6. Массив свободных областей памяти.
7. Сообщения о результате выполнения программы или ее подпрограмм.

Способ обращения к программе

Пользователь обращается к программе при помощи исполняемого файла app.exe.

Возможные аварийные ситуации и ошибки пользователя

Все возможные аварийные ситуации и ошибки пользователя описаны типом status_t, включающим в себя коды возврата программы:

1. ERR_IO: ошибка ввода/вывода данных программы.
2. ERR_RANGE: неверно указано количество чего-либо.
3. ERR_MEM: ошибка при работе с памятью.
4. ERR_OVERFLOW: ошибка переполнения стека.
5. ERR_POP_EMPTY: ошибка удаления элемента из пустого стека.
6. ERR_INVALID_POINTER: ошибка при работе с указателем.
7. ERR_ZERO_DIVISION: ошибка при попытке деления на ноль.

Описание внутренних структур данных

```
typedef struct
{
    int data[MAX_ARR_STACK_SIZE];
    size_t current_size;
} arr_stack_t;

typedef struct node
{
    int value;  // значение узла
    struct node *next; // указатель на следующий узел
} node_t;

typedef struct
{
    node_t *stack_pointer;  // указатель на вершину стека
    int curr_size;  // текущий размер стека
    int max_size;  // максимальный размер стека
} list_stack_t;

typedef struct
{
    void *free_blocks[MAX_FREE_BLOCKS_ARRAY_SIZE];
    size_t size;
} free_blocks_array_t;
```

Листинг 1. Внутренние структуры данных.

Структура arr_stack_t представляет собой стек, реализованный на основе статического массива. Хранит в себе текущий размер и значения стека.

Структура list_stack_t представляет собой стек, реализованный на основе динамического односвязного списка. Содержит в себе указатель на вершину стека node_t, текущий размер стека и максимальный размер стека.

Структура free_blocks_array_t – это структура для хранения массива свободных областей памяти, содержит в себе сам массив свободных областей типа void, а также размер этого массива.

Ниже представлены сигнатуры основных функций программы.

```
// операции для стека на основе статического массива
status_t push_arr_stack(arr_stack_t *arr_stack, int value);
status_t pop_arr_stack(arr_stack_t *arr_stack, int *popped_value);
status_t calc_arithmetic_expr_by_arr(const char *expression, int *result);

// операции для стека на основе динамического односвязного списка
status_t push_list_stack(list_stack_t *list_stack, int value);
status_t pop_list_stack(list_stack_t *list_stack, int *value, free_blocks_array_t
*free_blocks);
status_t free_list_stack(list_stack_t *list_stack);
status_t calc_arithmetic_expr_by_list(const char *expression, int *result,
free_blocks_array_t *free_blocks);

// операции для работы с массивом свободных областей памяти
status_t print_free_blocks(free_blocks_array_t *free_blocks);
status_t add_to_free_blocks_array(free_blocks_array_t *free_blocks, void *address);

// операции обработки текущей опции меню
```

```
status_t procces_menu_choice(int option, arr_stack_t *arr_stack, list_stack_t
*list_stack, free_blocks_array_t *free_blocks);

// операции сравнения производительностей и создания случайных выражений
status_t compare_pop_push(free_blocks_array_t *free_blocks);
status_t compare_performance(free_blocks_array_t *free_blocks);
status_t create_random_expression(char *expression, char operation);
```

Листинг 2. Сигнатуры основных функций.

Алгоритм

Помимо основного алгоритма программы (main.c) основополагающими алгоритмами для реализации заданного функционала являются:

- алгоритмы ввода данных (ввод с консоли, чтение из файла)
- алгоритм добавления элемента в стек (для массива и для массива)
- алгоритм удаления элемента из стека (для массива и для массива)
- алгоритм вычисления строкового выражения вида «число|знак|...число|знак|»
- алгоритм сравнения производительности операций pop и push для двух разных способов реализаций стека
- алгоритм сравнения производительности вычисления строкового арифметического выражения для двух разных способов реализаций стека

Рассмотрим каждый алгоритм.

```c
#include "../inc/data.h"
#include "arr_stack.h"
#include "input.h"
#include "output.h"
#include "process.h"
#include "free_blocks.h"


int main(void)
{
    status_t exit_code = SUCCESS_CODE;
    status_t menu_opt_processing_status = SUCCESS_CODE;
    menu_option_t cur_menu_opt = 0;  // выбранная опция меню

    arr_stack_t main_arr_stack = { { 0 }, 0 };
    list_stack_t main_list_stack = { NULL, 0, MAX_LIST_SIZE };
    free_blocks_array_t free_blocks = { { 0 }, 0 };

    print_menu();
    exit_code = input_cur_menu_opt(&cur_menu_opt);
    while (cur_menu_opt != 0 && exit_code == SUCCESS_CODE)
    {
        menu_opt_processing_status = procces_menu_choice(cur_menu_opt,
&main_arr_stack, &main_list_stack, &free_blocks);
        print_exit_code_result(menu_opt_processing_status);
        print_menu();
        exit_code = input_cur_menu_opt(&cur_menu_opt);
    }

    free_list_stack(&main_list_stack);

    print_exit_code_result(exit_code);

    return exit_code;
}
```

Листинг 3. Основной алгоритм программы.

1. Отображается меню и считывается выбранный пользователем пункт.
2. Инициализируются основные структуры: стек-массив, стек-список и массив свободных блоков.
3. Проверяется корректность ввода пункта меню.
4. Пока выбранная опция не равна нулю и нет ошибок, выполняется цикл.
5. Внутри цикла вызывается функция обработки выбранной опции.
6. Выводится результат выполнения обработки.
7. После выхода из цикла освобождаются ресурсы, выводится код завершения, и программа завершается.

Листинг 4. Основной алгоритм программы.

1. Инициализируется код возврата.
2. Проверяется корректность входных данных.
3. Выводится приглашение к вводу арифметического выражения.
4. Считывается строка с клавиатуры в заданный буфер.
5. Если ввод успешен, удаляется символ новой строки.
6. Если ввод не выполнен, устанавливается код ошибки ввода.
7. При отсутствии ошибок выводится введенное выражение.
8. Возвращается итоговый код возврата.

Листинг 5. Алгоритм ввода строкового арифметического выражения.

1. Инициализируется код возврата.
2. Проверяется корректность входных данных.
3. Если стек реализован как массив, и он заполнен фиксируется ошибка переполнения.
4. Если нет ошибок, создается новый элемент (в случае списка) или значение записывается в массив на текущую вершину.
5. Обновляется текущая вершина стека и увеличивается счетчик элементов.
6. Возвращается итоговый код возврата.

Листинг 6. Алгоритм добавления элемента в стек.

1. Инициализируется код возврата.
2. Проверяется корректность входных данных.
3. Проверяется, что стек не пуст.
4. Если стек реализован как список, удаляемый элемент временно сохраняется (для получения его значения в дальнейшем), вершина стека обновляется.
5. Если стек реализован как массив, уменьшается счётчик текущих элементов.
6. Если передан указатель для сохранения значения, то возвращается извлечённое значение.
7. Если стек реализован как список, освобождается память удалённого узла, добавляя его адрес в массив свободных блоков.
8. Возвращается итоговый код возврата.

Листинг 7. Алгоритм удаления элемента из стека.

1. Проверяется корректность входных данных, если некорректны – устанавливается соответствующий код возврата.
2. Инициализируются необходимые стеки для операндов и операторов (массив или список), а также вспомогательные переменные.
3. Если выражение пустое - устанавливается соответствующая ошибка.
4. Выражение посимвольно обрабатывается, выделяются операторы и операнды, помещаются в соответствующие стеки.
5. Из стека операндов извлекаются нужные значения, выполняется соответствующая операция, результат возвращается обратно в стек операндов. В конце вычислений тот единственный элемент, который остаётся в стеке операндов, и есть итоговый результат вычисления выражения.
6. Если на предыдущем этапе не возникло ошибок, извлекается финальный результат из стека операндов.
7. Если результат успешно получен, он помещается в выходную переменную.
8. При использовании стеков-списков освобождается вся выделенная память стеков после вычисления.
9. Возвращается итоговый код возврата.

Листинг 8. Алгоритм вычисления строкового выражения вида «число|знак|...число|знак|».

1. Инициализируются код возврата, необходимые структуры для двух видов стека, а также переменные для хранения времени выполнения операций.
2. Запускается замер времени добавления (push) элементов в стек-массив, каждую итерацию цикла выполняются операции добавления с замером времени.
3. Запускается замер времени удаления (pop) из стек-массива, каждую итерацию цикла выполняются операции удаления с замером времени.
4. Выполняются замеры времени добавления (push) и удаления (pop) для стек-списка.
5. Вычисляется среднее время одной операции для каждого типа стека и каждой операции.
6. Выводится таблица сравнительных результатов времени операций.

7. Освобождаются ресурсы, выделенные для стек-списка.
8. Возвращается итоговый код возврата.

Листинг 9. Алгоритм сравнения производительности операций pop и push.

1. Инициализируются код возврата, переменные для хранения выражения, операций, размеров стеков, времени выполнения операций и используемой памяти.
2. Для каждой арифметической операции в списке выполняется цикл по каждому размеру стека.
3. Для каждого размера стека рассчитываются среднее время вычисления выражения и объем памяти для массива и списка, повторяя вычисления несколько раз, чтобы усреднить результат.
4. Во вложенном цикле для массива генерируется случайное выражение, замеряется время его вычисления.
5. Во вложенном цикле для списка генерируется случайное выражение, замеряется время его вычисления на списке.
6. После завершения итераций вычисляются средние значения времени, используемой памяти для каждой структуры.
7. Результаты выводятся в строку таблиц).
8. Возвращается итоговый код возврата.

Листинг 10. Алгоритм сравнения производительности вычисления строкового арифметического выражения.

Тесты

Позитивные тесты

Номер теста	Описание	Ожидаемый результат
1	Добавление элемента «52» в стек (массив). Размер стека 0.	Стек содержит элемент «52». Размер стека 1.
2	Добавление элемента «52» в стек (массив): [1, 2, 3, 4, 5]. Размер стека 5.	Стек содержит следующие элементы: [1, 2, 3, 4, 5, 52]. Длина стека – 6.
3	Добавление элемента «1933» в стек (массив). Размер стека MAX_ARR_STACK_SIZE – 1.	Стек содержит MAX_ARR_STACK _SIZE элементов, последний из которых «52».
4	Добавление элемента «52» в стек (список). Размер стека 0.	Стек содержит элемент «52». Размер стека 1.
5	Добавление элемента «52» в стек (список): [1, 2, 3, 4, 5]. Размер стека 5.	Стек содержит следующие элементы: [1, 2, 3, 4, 5, 52]. Длина стека – 6.
6	Добавление элемента «1933» в стек (список). Размер стека MAX_ARR_STACK_SIZE – 1.	Стек содержит MAX_ARR_STACK _SIZE элементов, последний из которых «52».
7	Удаление элемента из непустого стека (массив).	Удаленный элемент стека: 52
8	Удаление элемента из непустого стека (список).	Удаленный элемент стека: 52

9	Вычисление выражения «1+5/3*6+1972» на основе стек-массива.	Введенное выражение: 1+5/3*6+1972 Результат вычислений: 1984
10	Вычисление выражения «1+5/3*6+1972» на основе стек-списка.	Введенное выражение: 1+5/3*6+1972 Результат вычислений: 1984
11	Сравнение эффективности операций push и pop.	Сравнительная таблица операций push и pop.
12	Сравнение эффективности вычисления выражения для двух способов реализации стека.	Таблица эффективности вычисления выражения для двух способов реализации стека.

Таблица 1. Позитивные тесты.

Негативные тесты

Номер теста	Описание	Ожидаемый результат
1	Попытка удаления элемента из пустого стек-массива.	Произошла попытка удаления элемента из пустого стека
2	Попытка удаления элемента из пустого стек-списка.	Произошла попытка удаления элемента из пустого стека
3	Попытка добавления элемента в заполненный стек-массив.	Введите элемент стека: 123 Произошло переполнение чего-нибудь
4	Попытка добавления некорректного элемента в стек-массив.	Введите элемент стека: milpopsjujuju Произошла ошибка ввода/вывода :(
5	Попытка добавления некорректного элемента в стек-список.	Введите элемент стека: milpopsjujuju Произошла ошибка ввода/вывода :(
6	Попытка деления на ноль в арифметическом выражении, вычисляемом на основе стек-массива.	Введите арифметическое выражение: 25 / 0 Введенное выражение: 25 / 0 Произошла попытка деления на 0
7	Попытка деления на ноль в арифметическом выражении, вычисляемом на основе стек-списка.	Введите арифметическое выражение: 25 / 0 Введенное выражение: 25 / 0 Произошла попытка деления на 0
8	Ввод некорректной опции меню.	Выберите пункт меню: pisyatdva Произошла ошибка ввода/вывода :(

Таблица 2. Негативные тесты.

Оценка эффективности

Сравним эффективность вычисления арифметического выражения на основе двух способов реализации стека. В таблице №3 представлены временные результаты вычисления выражения на стек-массиве и стек-списке для операций сложения, вычитания, деления и умножения. Значение каждой ячейки было получено как

среднее арифметическое из 1000 итераций. Время в таблице указано в наносекундах, объем памяти – в байтах.

Операция	Размер стека	Память (array)	Память (list)	Время (array)	Время (list)
+	10	4008	176	1528.92	1974.78
+	50	4008	816	5566.13	9295.12
+	100	4008	1616	10952.29	18404.08
+	250	4008	4016	25968.74	45220.52
+	500	4008	8016	51994.87	90621.68
+	1000	4008	16016	103525.59	185272.30
-	10	4008	176	1248.40	2056.14
-	50	4008	816	5984.64	9848.69
-	100	4008	1616	10713.54	19434.81
-	250	4008	4016	24446.14	46447.68
-	500	4008	8016	52288.19	92909.59
-	1000	4008	16016	101971.05	183517.71
/	10	4008	176	1259.74	1835.73
/	50	4008	816	5803.92	9344.43
/	100	4008	1616	11024.04	19453.03
/	250	4008	4016	26605.16	47260.19
/	500	4008	8016	54249.51	100016.98
/	1000	4008	16016	107027.90	187682.71
*	10	4008	176	1214.49	1751.94
*	50	4008	816	5685.40	10038.08
*	100	4008	1616	16918.46	20490.16
*	250	4008	4016	25813.81	45459.54
*	500	4008	8016	51585.93	91307.09
*	1000	4008	16016	103796.50	181107.44

Таблица 3. Эффективность вычисления выражения для двух способов реализации стека (наносекунды).

Исходя из таблицы №3 можно отметить, что разница реализаций стек-массива и стек-списка заключается в количестве памяти, потребляемой структурой и времени, за которое обрабатывается стек на основе каждой структуры. В контексте расходуемой памяти стек-список оказывается эффективнее, чем стек-массив только на маленьких выборках данных (до 250 элементов, то есть до 25% процентов от максимальной длины массива). Если стек насчитывает более 500 элементов, эффективность использования памяти стек-массива значительно превосходит способ реализации с помощью стек-списка. Также, согласно приведённой таблице, очевидно, что стек-массив работает быстрее, чем стек-список. Оценка отношения скорости вычисления выражения с помощью стек-списка по отношению к скорости вычисления выражения с помощью стек-массива представлена в таблице №4, в среднем стек-массив работает быстрее, чем стек-список в 1.854 раз. Для больших

объёмов данных стек-массив обеспечивает более эффективное использование памяти и значительно большую скорость работы по сравнению со стеком-списком, тогда как использование списка может быть оправдано только при работе с малыми данными.

Размер стека	+	-	/	*
10	1.29	1.64	1.45	1.44
50	1.67	1.64	1.61	1.77
100	1.68	1.81	1.76	1.21
250	1.74	1.9	1.78	1.76
500	1.74	1.78	1.84	1.77
1000	1.78	1.80	1.75	1.74

Таблица 4. Отношение времени выполнения арифметической операции для стек-списка ко времени выполнения арифметической операции для стек-массива.

На основании данных таблицы №5 можно заметить, что стек-массив выполняет операции «push» и «pop» значительно быстрее, чем стек-список. Это объясняется отсутствием необходимости выделения памяти и добавления нового узла в массивной реализации. Таким образом, массивный стек более эффективен для реализаций, в которых часто приходится производить операции добавления и удаления элементов стека. В таблице №5 приведено среднее арифметическое время выполнения операций pop и push (наносекунды), полученное из 1000 итераций.

Вид стека	«push»-операция	«pop»-операция
Стек-массив	50.58	43.17
Стек-список	124.23	55.21

Таблица 5. Сравнительная таблица операций push и pop для двух способов реализации стека (наносекунды).

В таблице №6 описывается результат следующей последовательности действий: сначала в стек-список добавляется 6 элементов (от 1 до 6), далее из него удаляются 3 элемента (6, 5, 4), в стек записываются элементы 7, 8, 9. Результаты выполнения данной последовательности действий, описанные в таблице №6, демонстрируют отсутствие фрагментации.

Текущий стек	Массив свободных областей памяти
Элемент списка №0: 9 (0xe974bd0) Элемент списка №1: 8 (0xe974bb0) Элемент списка №2: 7 (0xe974b90) Элемент списка №3: 3 (0xe974b70) Элемент списка №4: 2 (0xe974b50) Элемент списка №5: 1 (0xe974b30) Длина данного стека (списка): 6 Занимаемая память стека (списка): 112 байт(-ов)	Всего освобождено блоков: 3 [1] Адрес: 0xe974bd0 [2] Адрес: 0xe974bb0 [3] Адрес: 0xe974b90

Таблица 6. Фрагментация памяти при удалении и добавлении элементов в стек-список.

Вывод

Стек-массив демонстрирует явное преимущество перед стеком-списком как по скорости выполнения операций «push» и «pop», так и по эффективному расходу памяти при работе с объемными наборами данных. Стек-список оправдан только при небольшом количестве элементов, тогда как во всех остальных случаях массивная реализация обеспечивает более высокую производительность и экономичное использование ресурсов, что делает её предпочтительным выбором для решения большинства задач, требующих работы со стеком.

Таким образом, стек на массиве лучше подходит для задач с известным или ограниченным размером и высокими требованиями к скорост, а стек на списке предпочтительнее, если возможны большие или непредсказуемые изменения размера, как при динамической обработке потоковых данных.

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, Last In – First Out (LIFO).

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

   Существует несколько реализаций стека, в данной лабораторной работе рассматривается реализация стека с помощью одномерного статического массива и с помощью динамического односвязного списка.

   Для первого случая память выделяется только для одномерного статического массива. Размер массива фиксирован, вся память под стек выделяется один раз, одновременно.

   Для второго случая память выделяется динамически по мере необходимости для каждого нового элемента списка. Каждый такой элемент содержит свое значение и указатель на следующий элемент списка.

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

   При удалении элемента из стека, реализованного на основе одномерного статического массива, память не освобождается в привычном понимании «освобождения памяти», вместо этого длинна стека уменьшается на единицу, а указатель на вершину стека смещается. Область памяти, в

которой все еще хранится удаленный элемент стека, может быть перезаписана в дальнейшем, например, при добавлении нового элемента.

При удалении элемента из стека, реализованного на основе односвязного динамического списка, соответствующий узел списка удаляется, в память, выделенная под него, освобождается.

4. *Что происходит с элементами стека при его просмотре?*

   Обычно для просмотра стека используется опеарция pop: каждый элемент стека извлекается из него и выводится на экран, в результате чего в стеке не остается элементов.

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

   Выбор способа реализации стека зависит от нескольких факторов, таких как требования к скорости работы программы, частоте добавлений и удалений элементов стека, ограничению размера стека и т.п.

   Стек на основе одномерного статического массива отличается от реализации на списке высокой скоростью и простотой организации: накладные расходы при операциях push и pop минимальны, а также память выделяется один раз для всего массива. К недостаткам данного подхода можно отнести невозможность изменять его максимальный размер и нерациональное использование памяти (если используется только часть массива).

   В случае реализации стека на односвязном динамическом списке каждый элемент создаётся только когда это необходимо, то есть память под новые узлы выделяется по мере роста структуры. Однако суммарный расход памяти увеличивается из-за необходимости хранения указателей на узлы списка, а операции добавления и удаления из-за вызова динамического выделения и освобождения памяти выполняются медленнее, чем в массиве.