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 **Ширяев А.А.**

Группа **ИУ7-33Б**

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

Студент _____ **Ширяев А.А.**

Преподаватель _____**Силантьева. А.В**

2024

Цель работы — реализовать операции работы со стеком, который представлен в виде статического массива и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации; получить представление о механизмах выделения и освобождения памяти при работе со стеком.

Условие задачи

Вариант 4

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) статическим массивом (дополнительно можно реализовать динамическим 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.

Проверить правильность расстановки скобок трех типов (круглых, квадратных и фигурных) в выражении.

ПРИМЕЧАНИЕ! В отчёте будут использоваться обозначения для действий:

<init> - Инициализация выражения (строки)

<show_static> - Вывод статического стека на экран

<pop_static> - Удаление последнего элемента статического стека

<push_static> - Добавление элемента в конец статического стека

<clean_static> - Очистить статический стек

<show_list> - Вывод списка-стека на экран

<pop_list> - Удаление последнего элемента списка-стека

<push_list> - Добавление элемента в конец списка-стека

<clean_list> - Очистить список-стек

<check_static> - Проверить выражение на корректность, используя статический стек

<check_list>- Проверить выражение на корректность, используя список-стек

<popped_ad>- Вывод адресов освобождаемых элементов

<stat>- Вывод статистики

<exit> - Выход из программы

Описание техзадачи

1.Описание исходных данных

Данные на входе: Меню. Код действия. Далее для каждого действия:

<init> - Выражение в формате строки

<show_static> - Статический стек

<pop_static> - Статический стек

<push_static> - Статический стек, элемент для добавления

<clean_static> - Статический стек

<show_list> - Список-стек

<pop_list> - Список-стек, массив адресов освобождаемых элементов

<push_list> - Список-стек, элемент для добавления

<clean_list> - Список-стек

<check_static> - Статический стек, выражение в формате строки

<check_list>- Список-стек, выражение в формате строки, массив адресов освобождаемых элементов

<popped_ad>- Массив адресов освобождаемых элементов

<stat>- Статический стек, список-стек

<exit> - -

Данные на выходе:

<init> - Строка с выражением

<show_static> - Статический стек, выведенный на экран

<pop_static> - Статический стек, удалённый элемент

<push_static> - Статический стек

<clean_static> - Статический стек

<show_list> - Список-стек, выведенный на экран

<pop_list> - Список-стек, удалённый элемент

<push_list> - Список-стек

<clean_list> - Список-стек

<check_static> - Сообщение о результатах проверки

<check_list>- Сообщение о результатах проверки

<popped_ad>- Массив адресов освобождаемых элементов, выведенный на экран

<stat>- Таблица с временем и затраченной памятью на удаление/добавление элементов. Средние показатели времени удаления/добавления. Сравнение времени работы операций/памяти статического стека и списка-стека

<exit> - -

1.Описание задачи, реализуемой программой

Программа реализует ряд действий:

<init> - Инициализация выражения (строки)

<show_static> - Вывод статического стека на экран

<pop_static> - Удаление последнего элемента статического стека

<push_static> - Добавление элемента в конец статического стека

<clean_static> - Очистить статический стек

<show_list> - Вывод списка-стека на экран

<pop_list> - Удаление последнего элемента списка-стека

<push_list> - Добавление элемента в конец списка-стека

<clean_list> - Очистить список-стек

<check_static> - Проверить выражение на корректность, используя статический стек

<check_list>- Проверить выражение на корректность, используя список-стек

<popped_ad>- Вывод адресов освобождаемых элементов

<stat>- Вывод статистики

<exit> - Выход из программы

1.Способ обращения к программе

Для обращения к программе запускается файл app.exe.

1. Описание возможных аварийных ситуаций и ошибок пользователя

Программа может не вывести результат, а вывести сообщение об ошибке. Данная ситуация может произойти при условии:

Ошибки пользователя

1. Неверный код действия
2. <init> - Пустая строка
3. <pop_static> - Стек пуст
4. <push_static> - Стек переполнен
5. <pop_list> - Стек пуст
6. <push_list> - Память под элемент не смогла выделиться
7. <check_static> - Нет выражения
8. <check_list>- Нет выражения
9. <stat>- Вывод статистики
10. <exit> - Выход из программы

Аварийные ситуации

1. Программа не смогла выделить необходимую память для работы (с массивом освобождённых адресов)
2. Программа не смогла выделить необходимую память для работы (для добавления элемента в список-стек)

Описание внутренних СД

Информация о статическом стеке

```
#define STACK_MAX_SIZE 40
typedef struct
{
    char stack[STACK_MAX_SIZE];
    size_t size;
} static_stack_t;
```

Информация о статическом стеке представляет собой структуру на языке Си, состоящую из:

stack - Массив, хранящий элементы статического стека. Размер stack – текстовая замена STACK_MAX_SIZE

size — Текущий размер стека

Информация о списке-стеке:

```
typedef struct list_stack_t_ list_stack_t;

typedef struct list_stack_t_
{
    char value;

    list_stack_t *next;
} a_t;
```

Информация о списке-стеке представляет собой структуру на языке Си, состоящее из:

value – значение элемента

*next – указатель на предыдущий элемент

Информация о массиве освобождённых адресов:

```
typedef struct
{
    void **addresses;

    size_t size;
    size_t cap;
} addresses_t;
```

Информация о массиве освобождённых адресов представляет собой структуру на языке Си, состоящую из:

**addresses — Указатель на память, содержащую адреса освобождённых элементов

size – Размер массива

cap – Размер выделенной памяти под массив

Информация о выражении:

```
char *str = NULL;
size_t alloc_str = 0;
```

Информация о выражении на языке Си представляет собой:

*str — Указатель на строку

alloc_str — Размер выделенной под строку памяти

Описание алгоритма

Для начала считывается код действия. Далее в зависимости от выбранного действия:

<init>

- Выделяется минимальная память под строку, если она ещё не выделена

```
if (str == NULL)
{
    if ((str = malloc(sizeof(char))) == NULL)
    {
        printf("\nCOMPUTER CAN'T ALLOC MEMORY FOR STRING\n");

        break;
    }
    else
        alloc_str = sizeof(char);
}
```

- Вводится строка
- Под строку выделяется нужная память, если выделенной не хватает

```
printf("Enter string: ");
if (getline(&str, &alloc_str, stdin) == -1)
{
    printf("\nCOMPUTER CAN'T ALLOC MEMORY FOR STRING\n");

    string_free(&str, &alloc_str);
    break;
}
```

<show_static>

- Выводится статический стек

```
void static_stack_show(static_stack_t *stack)
```

<pop_static>

- Удаляется элемент

```
int static_stack_pop(char *data, static_stack_t *stack)
```

- Выводится информация о выведенном элементе

<push_static>

- Элемент добавляется в конец статического стека

```
int static_stack_push(char data, static_stack_t *stack)
```

<clean_static>

- Происходит удаление последнего элемента стека (<pop_static>) до того момента, пока стек не станет пустым

```
void static_stack_free(static_stack_t *stack)
```

<show_list>

- Выводится список-стек

```
void list_stack_show(list_stack_t *head)
```

<pop_list>

- Удаляется последний элемент.
- Выводится удалённый элемент с адресом освобождённой под элемент памяти
- Данные об удалённом элементе добавляются в массив освобождённых элементов

```
int list_stack_pop(char *c, list_stack_t **head)
```

<push_list>

- Выделяется память под новый элемент
- Элемент добавляется в конец списка-стека

```
int list_stack_push(char c, list_stack_t **head)
```

<clean_list>

- Происходит удаление последнего элемента стека (<pop_static>) (и освобождение его памяти) до того момента, пока стек не станет пустым

```
void list_stack_free(list_stack_t **head)
{
    char c;

    while (! list_stack_pop(&c, head));
}
```

<check_static>

- Происходит проход по строке.

Если перед нами открывающаяся скобка: Символ добавляется стек

Если перед нами закрывающаяся скобка: Удаляется символ из стека. Если скобки с одинаковыми типами

- Выводится результат в зависимости от стека: Если он пустой — CORRECT, иначе INCORRECT

```
int static_stack_string_check(char *str, static_stack_t *stack)
```

<check_static>

- Происходит проход по строке.

Если перед нами открывающаяся скобка: Символ добавляется стек

Если перед нами закрывающаяся скобка: Удаляется символ из стека. Если скобки с одинаковыми типами

- Выводится результат в зависимости от стека: Если он пустой — CORRECT, иначе INCORRECT

```
int list_stack_string_check(char *str, list_stack_t **head)
```

<popped_ad>

- Выводится массив освобождённых адресов

```
void addresses_show(void)
```

<stat>

```
void stack_statistics(static_stack_t *static_stack, list_stack_t **list_stack_head)
```

- Считается затраченная память на стеки всех типов
- Программа циклически выполняет N элементов добавляются и удаляются (порядок важен) ITER_COUNT раз.

```
#define ITER_COUNT 1000
```

- Высчитывается среднее значение времени
- Выводится строка с подсчитанными данными

<exit>

```
case CODE_EXIT:
    string_free(&str, &alloc_str);
    list_stack_free(&list_stack_head);
    addresses_free();

    printf("\nHave a nice day!\n");
    flag = false;

    break;
```

- Освобождается память, выделенная под динамически выделенные данные программы (массив освобождённых элементов и список-стек)
- Программа завершает свою работу

Набор тестов

В ходе выполнения лабораторной работы были написаны тесты для проверки работы программы

Позитивные тесты

Номер теста	Входные данные	Ожидаемые выходные данные
01 - <init> Ввод строки	1 (a + b) * (c + [d + {f}])	STRING WAS ENTERED SUCCESSFULLY
02 - <show_stack> Показ стека	1 {{ 11 2	{ {
03 - <pop_static> Удаление элемента	4 e 3	Popped: e
04 - <push_static> Добавление элемента	4 e 2	e
05 - <clean_ static> Удаление непустого стека	4 e 4 e	Stack (last to first):

	4 e 5 2	
06 - <show_list> Показ списка-стека с единственным элементом	8 X 6	X (address)
07 - <pop_list> Удаление одного из элементов	8 a 8 b 8 c 7	Popped: c (address)
08 - <push_list> Добавление нескольких элементов в список-стек	8 a 8 b 8 c 2	c (address) b (address) a (address)
09 - <clean_list> Удаление непустого списка-стека	8 a	Stack (last to first):

	8 b 8 c 9 2	
10 - <check_static> Проверка корректной строки	1 [()()()(){}] 10	Result: CORRECT
11 - <check_static> Проверка некорректной строки	1 [][][{]} 10	Result: INCORRECT
12 - <check_list> Проверка корректной строки	1 {[()()()(){}]}{}{00000}({}) 11	Result: CORRECT
13 - <check_list> Проверка некорректной строки	1 []([000)] 11	Result: INCORRECT
14 - <popped_ad> 3 удалённых адреса	8 a 8 b	(address_c) (address_b) (address_a)

	8 с 12	
15 - <stat> Вывод статистики по операциям со стеками	13	Вывод таблицы с результатами подсчётов времени и
16 - <exit> Выход из программы	14	-

Негативные тесты

Номер теста	Входные данные	Выходные данные
01 - Код неправильный	15	INVALID ENTERED DATA
02 - Код с иными символами	1.1	INVALID ENTERED DATA
03 - <init> Строка пустая	1	STRING CAN'T BE EMPTY
04 - <pop_static> Нет элементов	3	STACK IS EMPTY
05 - <pop_list> Нет элементов	7	STACK IS EMPTY
06 - <check_static> Выражение отсутствует	10	NO DATA
07 - <check_list> Выражение отсутствует	11	NO DATA
08 - <push_static> переполнение стека (Предварительно заменим значение текстовой	8	STACK IS OVERFLOWED

замены MAX_STACK_SIZE значением 1)	{ 8 r	

Таблицы с результатами измерений времени и памяти при выполнении различных операций со статическим стеком и со списком-стеком

Ниже представлены результаты статистики при различных количествах элементов (для получения значения использовалось 1000 итераций (Значение ITER_COUNT))

```
STATISTICS (time in nsec) (memory in bytes) (STATIC STACK SIZE IS 40000)
N count |PUSH t (static) | POP t (static) |  PUSH t (list) |   POP t (list) |Memory (static) | Memory (list)  |  best t (PUSH) |  best t (POP)  |  best memory   |
2       |50              |53              |54              |60              |40008           |40              |     STATIC     |     STATIC     |      LIST      |
4       |57              |63              |67              |86              |40008           |72              |     STATIC     |     STATIC     |      LIST      |
8       |92              |101             |122             |153             |40008           |136             |     STATIC     |     STATIC     |      LIST      |
16      |127             |146             |226             |277             |40008           |264             |     STATIC     |     STATIC     |      LIST      |
32      |201             |243             |445             |457             |40008           |520             |     STATIC     |     STATIC     |      LIST      |
64      |353             |437             |880             |851             |40008           |1032            |     STATIC     |     STATIC     |      LIST      |
128     |651             |824             |1684            |1630            |40008           |2056            |     STATIC     |     STATIC     |      LIST      |
256     |1318            |1663            |3349            |3227            |40008           |4104            |     STATIC     |     STATIC     |      LIST      |
512     |2307            |2953            |4510            |4398            |40008           |8200            |     STATIC     |     STATIC     |      LIST      |
1024    |3447            |4582            |9768            |9422            |40008           |16392           |     STATIC     |     STATIC     |      LIST      |
2048    |7756            |10480           |18360           |17238           |40008           |32776           |     STATIC     |     STATIC     |      LIST      |
4096    |12495           |16078           |40954           |34617           |40008           |65544           |     STATIC     |     STATIC     |     STATIC     |
8192    |30393           |38857           |88262           |72797           |40008           |131080          |     STATIC     |     STATIC     |     STATIC     |
16384   |52358           |67802           |158563          |128908          |40008           |262152          |     STATIC     |     STATIC     |     STATIC     |
32768   |108397          |133454          |324963          |273962          |40008           |524296          |     STATIC     |     STATIC     |     STATIC     |

Average push time (static): 3

Average pop time (static): 4

Average push time (list): 9

Average pop time (list): 8
```

На основе полученных данных можно сделать следующие выводы:

- При работе с маленьким количеством элементов стека список-стек будет выгоднее по памяти, но при большом количестве элементов список-стек будет проигрывать статическому стеку по памяти.
- Вне зависимости от количества элементов PUSH и POP статического стека работают быстрее PUSH и POP списка-стека: среднее время PUSH статического стека быстрее среднего времени PUSH списка-стека в 3 раза или в 300% (9 nsec / 3 nsec = 3); среднее время POP статического стека быстрее среднего времени POP списка-стека в 2 раза или в 200% (8 nsec / 4 nsec = 2).
- С возрастанием количества добавленных/удалённых элементов скорость операций увеличивается, но с ~2048 элементов скорость нормализуется и колеблется в окрестности среднего значения.

Выводы по проделанной работе

Если нельзя предположить, с каким объёмом данных будет работать программа, стоит реализовать стек как связный список: данный вариант ограничивается лишь аппаратными возможностями компьютера. В противном случае стоит реализовать стек через статический массив: скорость операций при статическом массиве в разы быстрее, чем операции при связном списке.

(ПРИМЕЧАНИЕ! В ходе работы программы при ручном добавлении(PUSH) и удалении(POP) фрагментации не наблюдалось, но после подсчёта статистики фрагментация наблюдалась)

Контрольные вопросы

1. Что такое стек?

Стек — хранилище данных, работающее по принципу «LIFO», «last in first out» или «Последний пришёл — Первый ушёл»

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

Статический массив — память выделяется сразу под максимальное количество элементов. Размер такого стека известен ещё на этапе компиляции и никаким образом не может быть изменён.

Связный список — память динамически выделяется постепенно: с возрастанием количества элементов в стеке.

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

Статический массив — память под стек постоянна и не изменяется в процессе работы программы.

Связный список — память удаляется динамически

4. Что происходит с элементами стека при его просмотре?

Классическая реализация стека подразумевает, что просмотр элементов происходит только извлечением элементов из стека.

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

Реализация стека зависит от нескольких основных факторов:

1. Скорость работы операций

2. Размер предполагаемых данных

Опираясь на статистику, эффективнее реализовывать стек через массив: скорость его работы в разы превышает скорость работы операций со списком, а при большом количестве данных массив и вовсе превосходит список по всем параметрам, но для использования массива необходимо знать объём обрабатываемых данных.

Если же мы не знаем объём обрабатываемых данных, стоит использовать связный список: в таком случае размер данных ограничен лишь возможностями компьютера.