Лабораторная работа №4. Работа со стеком.

Студент Варин Дмитрий Владимирович
Группа ИУ7-36Б

Цель

1. Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного линейного списка;
2. Оценить преимущества и недостатки каждой реализации: получить представление о механизмах выделения и освобождения памяти при работе со стеком.

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**:

1. Возможность просмотра адресов элементов стека;
2. Создания дополнительного списка свободных областей (адресов освобождаемой памяти при удалении элемента, 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.

Задание варианта:

Элементами стека являются слова. Распечатайте слова в обратном порядке.

Техническое задание

Входные данные:

1. **Целое число - номер команды** :

- Число от 0 до 15.

2. **Данные, зависимые от команды**:

- **1** - добавить строку в *стек-список* - **пользователь вводит строку**;
- **2** - добавить строку в *стек-массив* - **пользователь вводит строку**;
- **12** - заполнить *стек-список* N случайными элементами - **пользователь целое число N**;
- **13** - заполнить *стек-массив* N случайными элементами - **пользователь целое число N**;

- **14** - удалить из *стека-списка* N элементов - **пользователь целое число N**;
- **15** - удалить из *стека-массива* N элементов - **пользователь целое число N**;

Выходные данные:

1. **Данные, зависимые от команды** :

- **0** - сообщение о завершении выполнения программы;
- **1** и **2** - сообщение о добавлении строки в стек список / массив;
- **3** и **4** - удалённая строка с вершины стека списка / массива;
- **5** и **6** - строка, находящаяся на вершине стека списка / массива;
- **7** и **8** строка, находящаяся на вершине стека списка / массива;
- **9** и **10** - сообщение об удалении элемента стека списка / массива;
- **11** - список свободных адресов для стека - списка;
- **12** и **13** - время заполнения стека списка / массива N элементами;
- **14** и **15** - время удаления N элементов из стека списка / массива;

Функции программы:

В зависимости от команды выполняются следующие действия :

- **0** - закончить выполнение программы;
- **1** - добавить строку в *стек-список*;
- **2** - добавить строку в *стек-массив*;
- **3** - удалить элемент с вершины *стека-списка*;
- **4** - удалить элемент с вершины *стека-массива*;
- **5** - посмотреть строку на вершине *стека-списка*;
- **6** - посмотреть строку на вершине *стека-массива*;
- **7** - вывести *стек-список*;
- **8** - вывести *стек-массив*;
- **9** - очистить *стек-список*;
- **10** - очистить *стек-массив*;
- **11** - вывести **список свободных адресов** для *стека-списка*;
- **12** - заполнить *стек-список* N случайными элементами **с замером времени**;
- **13** - заполнить *стек-массив* N случайными элементами **с замером времени**;
- **14** - удалить из *стека-списка* N элементов **с замером времени**;
- **15** - удалить из *стека-массива* N элементов **с замером времени**;

Обращение к программе:

Программа запускается из терминала;

Аварийные ситуации:

1. Некорректный ввод номера команды.
   На входе: число, не входящее в диапазон команд.
   На выходе: сообщение «Неверный режим, повторите попытку...»
2. Ошибка выделения памяти под строку.
   На входе: строка.

На выходе : сообщение «Ошибка считывания строки, переводим в меню...»
3. Ошибка выделения памяти под структуры данных. *На входе* : *в зависимости от номера программы строка / число*.
   На выходе : сообщение «Ошибка выделения памяти, переводим в меню...»
4. Удаление элемента из пустого стека. *На входе* : целое число - номер команды.
   На выходе : сообщение «Стек пуст.»
5. Вывод элементов пустого стека.
   На входе : целое число - номер команды.
   На выходе : сообщение «Стек пуст.»
6. Просмотр элемента на вершине пустого стека.
   На входе : целое число - номер команды.
   На выходе : сообщение «Стек пуст.»
7. Удаление пустого стека.
   На входе : целое число - номер команды.
   На выходе : сообщение «Стек пуст.»
8. Ошибка заполнения стека случайными строками.
   На входе : целое число - номер команды.
   На выходе : сообщение «Ошибка добавления случайных строк...»
9. Удаление из пустого стека N элементов.
   На входе : целое число - номер команды.
   На выходе : сообщение «Сначала нужно добавить элементы».

Структуры данных:

Для представления стека использованы 2 АТД:

- односвязный список;
- динамически-расширяемый массив (вектор);

Односвязный список состоит из узлов, представленных структурой `node_t`, а также отдельной структурой для вершины `node_t top`.

```
typedef struct node_s
{
    char *data;
    struct node_s *next;
} node_t;

node_t *top;
```

`char *data` - строка узла.
`struct node_s *next` - указатель на следующий узел.

Динамически-расширяемый массив (вектор) представлен структурой `stack_arr_t`.

```
typedef struct
{
```

```
    char **data;
    int top;
    int capacity;
} stack_arr_t;
```

`char **data` - массив строк.
`int top` - вершина стека.
`int capacity` - вместимость вектора.

Для хранения адресов свободных областей памяти используется структура `free_addr_t`:

```
typedef struct
{
    node_t **free_ptr;
    size_t capacity;
    size_t len;
} free_addr_t;
```

`node_t **free_ptr` - массив указателей на свободные области памяти элементов стек-списка;
`size_t capacity` - вместимость массива;
`size_t len` - длина(количество элементов в массиве);
За длину стека отвечает:

```
#define STACK_LIM 5
```

По умолчанию установлено значение 5 для удобства отслеживания заполнения стека, при необходимости можно заменить на другую велиину.

Для взаимодействия со стеком реализованы функции:
Для работы с стек-списком:

```
int push(char *data, free_addr_t *list_addr);
int pop(free_addr_t *list_addr);
int peek();
int print_all();
int clean_stack_list(free_addr_t *list_addr);
void print_free_addr(free_addr_t *list_addr);
void clean_free_addr(free_addr_t *list_addr);
```

Для работы с стек-массивом:

```
stack_arr_t *create_s_arr();
int free_s_arr(stack_arr_t **arr);
int clean_s_arr(stack_arr_t *arr);
```

```
int push_s_arr(stack_arr_t **arr, char *str);
int pop_s_arr(stack_arr_t *arr);
int peek_s_arr(stack_arr_t *arr);
int print_s_arr(stack_arr_t *arr);
```

Алгоритм

1. Пользователь вводит номер команды из меню.
2. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия в соответствии с ТЗ.
3. В зависиомости от действия происходит добавление / удаление / вывод строк в (из) стек(а).
4. При выборе добавления / удаления N эл - в, происходит замер времени выполнения операций.

Оценка эффективности

Измерения времени умножения будут производиться в *микросекундах*.
Для измерения используется структура `timeval` из библиотеки `<sys/time.h>`.
При записи результатов использовалось среднее время, полученное по результатам добавления / удаления N элементов (строк).
Для подсчёта времени используются следующие функции :

```
size_t generate_random_len();
int rand_push(stack_arr_t **arr, free_addr_t **addresses, int flag);
int multi_pop(stack_arr_t *arr, free_addr_t *arr_free_addr, int flag);
```

Добавление строк в стек - сравнение времени

Количество	Стек - список	Стек - массив
100	91	109
1000	1286	846
10000	9583	8799
100000	88858	86102
1000000	902815	872449

Расход памяти добавление строк из 256 байт

Количество	Стек - список	Стек-массив
100	14508	12916
1000	145008	129016
10000	1450008	1290016
100000	14500008	12900016

Количество	Стек - список	Стек-массив
1000000	145000008	129000016

Удаление строк - замеры времени

Количество	Стек - список	Стек-массив
100	134	74
1000	930	702
10000	9165	7539
100000	78745	71184
1000000	750257	742810

Контрольные вопросы

1. **Что такое стек?**
   Стек - абстрактный тип данных, представляющий собой список элементов, организованных по принципу *LIFO*, причём действия можно производить только с последним элементом. *(last in first out)*.

2. **Каким образом и сколько памяти выделяется под хранение стека при различной его реализации?**
   Реализовать стек можно через односвязный список, либо через динамически-расширяемый (вектор).
   При реализации через список память выделяется на каждый узел, содержащий данные и указатель на следующий элемент стека (4 или 8 байт, в зависимости от архитектуры), также нужно выделить память для хранения указатель на вершину стека (4 или 8 байт, в зависимости от архитектуры).
   При реализации через вектор, выделяется память для данных, переменную для хранения индекса, и вместимость вектора.

3. **Каким образом освобождается память при удалении элемента стека при различной реализации стека?**
   При реализации через список, удаляя элемент, освобождается память из под узла.
   При реализации через вектор, смещается вершина стека, удаляется элемент.

4. **Что происходит с элементами стека при его просмотре?**
   При реализации стека через список просматриваются все узлы с верхнего до нижнего, если задача стоит просмотреть все.
   Есть возможность не удалять элементы, но из определения стека исходит, что после просмотра элемента, его следует удалить, т.к доступ можно получить только к элементу, находящемуся на вершине стека.

5. **Каким образом эффективнее реализовывать стек? От чего это зависит?** Стек эффективнее реализовывать через расширяемый массив, т.к :

- динамически-расширяемый массив затрачивает меньше памяти(не нужно хранить указатель на следующий эл-т, т.е экономия от 4 до 8 байт с элемента.);
- по времени обработка происходит быстрее массива, чем списка.