Лабораторная работа №4. "Работа со стеком"

Студент Ларин Владимир - ИУ7-34Б

Описание условия задачи

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

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека. Реализовать стек:

- массивом;
- 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.

Элементами стека являются адреса памяти. При реализации массивами -их вводить, при реализации списком –брать адрес выделенной памяти под элемент.

Техническое задание

Входные данные:

1. Целое число, представляющее собой номер команды: целое число в диапазоне от 0 до 7.
2. Командно-зависимые данные:
    - адрес памяти в формате `0xHEX`, где `HEX` - число в 16-тиричной системе счисления

Выходные данные:

1. Состояние стеков
2. Количественная характеристика сравнения вариантов обработки стека.

Функции программы

0. Выход

- Стек - массив

1. Распечатать весь стек
2. Добавить элемент в стек
3. Вытащить элемент из стека

- Стек - список

4. Распечатать весь стек
5. Добавить элемент в стек
6. Вытащить элемент из стека

- Сравнения

7. Провести анализ

Обращение к программе: запускается из терминала.

Аварийные ситуации

1. Некорректный ввод номера команды.
    - На входе: число не лежащее в отрезке [0; 7]
    - На выходе: сообщение об ошибке

2. Попытка извлечь элемент из пустого стека.
    - На входе: пустой стек
    - На выходе: сообщение об ошибке
3. Некорректный ввод адреса.
    - На входе: адрес, не соответсвующий ТЗ
    - На выходе: сообщение об ошибке

Структуры данных

Для хренения элементов стека использую псевдоним `stack_type_t` типа данных `void *`

```
typedef void* stack_type_t;
```

Для хранения стека на основ е массива использую структуру данных `array_stack_t`

```
typedef struct array_stack {
  size_t capacity; // Колличество выделенных элементов
  stack_type_t* end; // Верх стека
  stack_type_t* buffer; // Массив
} array_stack_t;
```

Для односвязного списка использую структуру `node_t`

```
typedef struct node {
  struct node* next; // Следующий элемент
  stack_type_t value; // Значение
} node_t;
```

Для хренения стека на основе односвязного списка использую структуру `list_stack_t`

```
typedef struct list_stack {
  node_t* list; // Односвязный список
  node_t* free_nodes; // Список свободных областей памяти
} list_stack_t;
```

Алгоритм

1. Пользователю выводится меню
2. Пользователь вводит номер команды
3. Выполняется действие согласно номеру команды

- Стек - массив
    1. Распечатать весь стек
    - Проходится по всем элементам массива и выводит их
    2. Добавить элемент в стек
    - Если массив заполнен полностью, то он расширяется вдвое
    - Добавляется новый элемент в массив
    3. Вытащить элемент из стека
    - Выводится последний элемент массива
    - В дискипторе уменьшается длинна стека
- Стек - список 4. Распечатать весь стек
    - Пробегается по всем элеметам списка и выводит их, при этом печатает их
    - Выводит свободные области оперативной памяти
    5. Добавить элемент в стек

    - Создается узел, указывающий на голову списка
    - В дескрипторе указатель на голову перенаправляют на новосозданный узел
  6. Вытащить элемент из стека
    - Выводится значение головы
    - Запоминается адрес головы
    - В дескрипторе указатель на голову перенаправляют на следующий (второй) элемент списка
    - Высвобождается старая голова
- Сравнения 7. Провести анализ
    - Перебираются различные длинны стека
    - Создаются новые стеки для каждого измерения
    - Выводится таблица с численными и объёмными характеристиками вставки, удаления элементов в стеках разных видов.

Описание функций

- Стек на основе массива:
    - `array_stack_t* init_array_stack();` - Создает стек
    - `int push_array_stack(array_stack_t* stack, stack_type_t element);` - добавляет элемент в стек
    - `int pop_array_stack(array_stack_t* stack, stack_type_t* element);` - вытаскивает элемент из стека
    - `void free_array_stack(array_stack_t* stack);` - Уничтожает весь стек
- Стек на основе списка:
    - `list_stack_t* init_list_stack();` - Создает стек
    - `node_t* create_node();` - Создает узел списка
    - `int push_list_stack(list_stack_t* stack);` - Добавляет элемент в стек
    - `int pop_list_stack(list_stack_t* stack, stack_type_t* element);` - Вытаскивает элемент из стека
    - `void free_list_stack(list_stack_t* stack);` - Уничтожает весь стек

Тесты

Меню

```
===== Стек - массив
  1) Распечатать весь стек
  2) Добавить элемент в стек
  3) Вытащить элемент из стека
===== Стек - список
  4) Распечатать весь стек
  5) Добавить элемент в стек
  6) Вытащить элемент из стека
===== Сравнения
  7) Провести анализ

  0) Выход
```

#	Описание теста	Ввод	Вывод
1	Добавление элементов в стек - массив	2 enter 0x345 enter enter 2 enter 0x333 enter enter 3 enter 3 enter	0x333 0x345
2	Добавление элементов в стек - список	5 enter 5 enter 6 enter 6 enter 4 enter	Пустой стек с двумя нодами в списке свободных элементов

#	Описание теста	Ввод	Вывод
3	Провести анализ	7 enter	Таблица с временем заполения и удаления стеков разной длины
4	Вытащить элемент из пустого стека - массива	3 enter	Сообщение об ошибке
5	Вытащить элемент из пустого стека - списка	6 enter	Сообщение об ошибке
7	Ввод неверного адресса	2 enter incorrect	Сообщение об ошибке

Оценка эффективности

Для каждого измерения взято среднее из 100 попыток измерения.

- Время указано в тактах процессора
- Объём указан в Байтах

В следующей теблице используется указатель в качестве содержимого.

```
typedef struct void* stack_type_t;
```

Длинна стека	Время добавления в стек, реализованный с помощью массива	Время добавления в стек, реализованный с помощью списка	Время удаления из стека, реализованного с помощью массива	Время удаления из стека, реализованного с помощью списка	Объём стека, реализованного с помощью массива	Объём стека, реализованного с помощью списка
2	2	2	2	2	40	48
16	1	1	0	0	152	272
128	1	3	1	1	1048	2064
1024	18	40	6	9	8216	16400
8192	104	216	40	59	65560	131088
65536	864	1732	320	446	524312	1048592
524288	5446	11808	2518	3861	4194328	8388624
3	0	0	0	0	56	64
17	0	1	0	0	280	288
129	2	3	1	1	2072	2080
1025	15	34	6	8	16408	16416
8193	124	269	44	61	131096	131104
65537	861	1712	322	447	1048600	1048608
524289	7714	13683	2575	3972	8388632	8388640

Так в размер узла списка превышает размер элемента ровно в два раза, объёмные характеристики при заполнености $N^2 + 1$, сравнимы с размером массива. Для эксперемента поменяем тип элемента стека с указателя (в моем случае 8 байт), на строку с максимальной длинной (256 - 8) = 248 Байт

```

```

```
typedef struct {
    char array[248];
} stack_type_t;
```

Длинна стека	Время добавления в стек, реализованный с помощью массива	Время добавления в стек, реализованный с помощью списка	Время удаления из стека, реализованного с помощью массива	Время удаления из стека, реализованного с помощью списка	Объём стека, реализованного с помощью массива	Объём стека, реализованного с помощью списка
4	3	3	2	3	1044	1072
16	4	4	2	3	4104	4240
64	6	6	3	4	16344	16912
256	18	19	9	19	65304	67600
1024	107	130	30	67	261144	270352
4096	510	511	117	264	1044504	1081360
16384	2242	2355	564	1404	4177944	4325392
65536	9083	10246	2274	5838	66846744	17301520
262144	28984	30778	8915	23204	66846744	69206032
5	1	1	0	1	2064	1336
17	1	1	1	1	8184	4504
65	4	4	2	3	32664	17176
257	15	15	7	12	130584	67864
1025	55	55	31	49	522264	270616
4097	210	215	125	207	2088984	1081624
16385	868	884	580	1024	8355864	4325656
65537	3483	3645	2299	4573	33423384	17301784
262145	53233	54302	8934	23304	133693464	69206296

Контрольные вопросы

1. Что такое стек?

   Стек – структура данных, в которой предусмотрена только обрабатка последнего элемент (верхний элемент). На стек действует правило - последний вошел, первым вышел.

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

   - Для каждого элемента стека, реализованного списком, выделяется память для хранения указателя и содержания элемента.
   - Для каждого элемента стека, реализованного массивом, выделяется память только для хранения содержания элемента.

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

   - При хранении стека связанным списком, верхний элемент удаляется освобождением памяти для него и смещением указателя, указывающего на начало стека.
   - При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека.

4. Что происходит с элементами стека при его просмотре?

- Все элементы стека удаляются, так как каждый раз достается верхний элемент стека.

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

    - Если колличество элементов заранее известно, и размер узла списка не превышат в два раза размер содержимого, то эффективнее использовать массив. Иначе при нехватке пямяти целесообразно использовать список.

Вывод

При указателе в качестве содержимого ($V_{context} \approx V_{ptr}$) и длинне стека 8192 (массив полность заполнен) стек-массив быстрее в два раза и менее затратен в памяти также в два раза, но при длине стека $8192 + 1 = 8193$ (массив наполовину пуст) объёмная характеристика стеков сравнима.

А в случае, когда в качестве содержимого строка 248 символов ($V_{context} >> V_{ptr}$), то при размере стека $65 536$ (массив полностью заполнен) размерная характеристика сравнима, скорость добавления сравнима, а скорость удаления у массивов в два раза больше. Но при размере стеека $65 536 + 1 = 65 537$ (массив наполовину пуст) стек-список занимает памяти почти в два раза меньше, чем стек-массив.

Делаем выводы:

- Использование стека, построеного СД список, целесообразно, когда размер указателя на следующий элемент намного меньше, чем размер содержимого узла, т.е. $V_{ptr} << V_{context}$.
- Также возможно использование стека-списка для экономии оперативной памяти в случаях, когда размер стека заранее не известен.
- В остальных случаях, т.е. когда максимальный размер стека заранее известен или когда $V_{ptr} >= V_{context}$, целесообразней использовать стек, построенный на СД массив.