# Работа со стеком

### Цель работы

### Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного списка, оценить преимущества и недостатки каждой реализации, получить представление о механизмах выделения и освобождения памяти при работе с динамическими структурами данных.

**Входные данные:**

Последовательности текстовых символов(слова), длина слова не более 19 знаков.

**Выходные данные:**

Последовательность слов, в порядке, обратном порядку ввода.

**Описание задачи:**

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) 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 свободных областей (адресов освобождаемых элементов) с выводом его на экран.

**Способ обращения к программе:**

Взаимодействие с программой происходит посредством командной строки

**Описание возможных аварийных ситуаций:**

* Длина слова более 19 знаков-выводится ошибка

**Сравнение по времени выполнения:**

В стек были последовательно добавлены элементы, а после этого удалены.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество элементов | Структура | Время | Выигрыш по времени |
| 10 | Список | **6.300000e-003 ms** | **-** |
| 5 | Список | **3.600000e-003 ms** | **-** |
| 5 | Массив | **3.000000e-004 ms** | **1100%** |
| 10 | Массив | **4.000000e-004 ms** | **1475%** |

**Сравнение по памяти**

|  |  |  |  |
| --- | --- | --- | --- |
| Количество элементов | Структура | Память(байт) | Выигрыш по памяти |
| 10 | Список | **240** | **-** |
| 5 | Список | **120** | **67%** |
| 5 | Массив | **200** | **-** |
| 10 | Массив | **200** | **20%** |

Использования массива для реализации стека дает значительный выигрыш в быстродействии, однако, если число элементов стека неизвестно заранее, то проигрывает в памяти. Так же была замечена фрагментация при удалении/добавлении элементов, то есть новый элемент не всегда запишется по тому же адресу, по которому мы освободили элемент.

**Внутренняя структура данных**

**Для реализации списком**

**struct** Node {

**char** **value**[20];

**struct** Node \***next**;

} ;

**Node** \*list=0; - указатель вершин стека

**value –** слово.

**\*next** – указатель на следующий элемент списка.

**Для реализации массивом**

**char** \*p – указатель вершины стека

**char** b[10][20]- массив для хранения слова

**Функции работы со стеком, реализация- список.**

**Извлечение элемента:**

**template** < **typename** T>

**void** pop2(T\*\* head) {

T \*out;

**if** (\*head == **NULL**) {

exit(-1);

}

out = \*head;

\*head = (\*head)->next;

free(out);

**return** ;

}

**Добавление элемента:**

**void** push(Node \*\*head,**const** **char** val[]) {

Node \*tmp =(Node \*) malloc(**sizeof**(Node));

**if** (tmp == **NULL**) { exit(-1) ; }

tmp->**next** = \*head;

strcpy(tmp->**value**,val);

\*head = tmp;

}

**Функции работы со стеком, реализация- массив**

**Добавление элемента**

**void** add\_array(**char** h[])

{

**if**(p<b[0]) {cout<<**"Stack** **is** **empty** **now"**;p=b[0];}

**if** (p<=(b[9]))

{

strcpy(p,h);

p=p+**sizeof**(b[0]);

}

**else**

{

cout<<**"\nStack** **is** **full** **now"**<<endl;

}

**return**;}

**Удаление элемента**

**void** del\_array()

{

**if** (p>=b[0])

{

p-=**sizeof**(b[0]);

}

**else**

cout<<"\nStack is empty now\n";

**return** ;

}