Отчёт

Лабораторная работа 4 по предмету «Типы и структуры данных».
Берёзкин Алексей, ИУ7-31Б, Вариант 6.

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.

Техническое задание

Задача

1. Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека.
2. Реализовать стек:
    1. массивом;
    2. списком.
3. При реализации стека списком в вывод текущего состояния стека добавить просмотр адресов освобождённых элементов.
4. Используя стек, определить, является ли строка палиндромом.

Входные данные

1. Тип стека.
2. Команда (добавление / удаление / определить, является ли палиндромом / статистика)

Выходные данные

1. Вывод текущего состояния стека.
2. Вывод адресов освобождённых элементов.
3. Вывод результата проверки строки на палиндром.

Аварийные ситуации

1. Некорректный ввод. Будет выведено сообщение об ошибке.
2. Попытка добавления в заполненный массив. Будет выведено сообщение об ошибке.
3. Попытка удаления или определения палиндрома на пустом стеке. Будет выведено сообщение об ошибке.

Структуры данных

Интерфейс стека

```
template <typename T>
struct IStack {
public:
    IStack(const size_t limit = LIMIT);
    virtual ~IStack() { }

    virtual void push(const T data) = 0;
    virtual T pop() = 0;
    virtual T top() const = 0;

    virtual void clear() = 0;

    size_t size() const;
    bool is_empty() const;
    bool is_full() const;

protected:
    size_t size_;
    size_t limit_;

    static const size_t STD_LIMIT = 1024;
};
```

Стек на массиве

```
template <typename T>
struct StackArray : public IStack<T> {
public:
    StackArray(const size_t limit = IStack<T>::STD_LIMIT);
    ~StackArray();

    void push(const T data) override;
    T pop() override;
    T top() const override;

    void clear() override;

protected:
    T *array_;
};
```

Стек на односвязном списке

```
template <typename T>
struct Node {
    T data;
    Node<T> *next;
```

```
    Node(T data = T());
};

template <typename T>
struct StackLinkedList : public IStack<T> {
public:
    StackLinkedList();
    ~StackLinkedList();

    void push(T data) override;
    T pop() override;
    T top() const override;

    void clear() override;

protected:
    Node<T> *head_;
    Node<T> *tail_;

    Node<T> *get_node(const size_t index); // const
};
```

Алгоритмы

Push

```
template <typename T>
void StackArray<T>::push(T data) {
    assert(!IStack<T>::full());

    array_[this->size_++] = data;
}

template <typename T>
void StackLinkedList<T>::push(T data) {
    Node<T> *node = new Node<T>(data);

    if (IStack<T>::size_++)
        tail_ = tail_->next = node;
    else
        tail_ = head_ = node;
}
```

Pop

```
template <typename T>
T StackArray<T>::pop() {
    assert(!IStack<T>::empty());

    return array_[--this->size_];
}

template <typename T>
T StackLinkedList<T>::pop() {
```

```
    assert(!IStack<T>::empty());

    T ret = tail_->data;
    if (IStack<T>::size_ == 1) {
        delete tail_;
        head_ = tail_ = nullptr;
    }
    else {
        Node<T> *pretail_ = get_node(IStack<T>::size_ - 2);
        delete tail_;
        pretail_->next = nullptr;
        tail_ = pretail_;
    }

    --IStack<T>::size_;

    return ret;
}
```

Palindrom

Половину стека, `pop` -ая, `push` -у во второй стек. Если изначально в стеке нечётное количество элементов, то удаляю ещё один элемент в стеке (который был в середине). Далее, пока два стека равной длины не опустошатся, `pop` -аю элементы из их вершин и сравниваю их. Если хоть раз не совпадут — не палиндром.

Тесты

```
palindrom ?

<empty> --- error
g --- yes
g 6 --- no
g 6 6 --- no
g 6 6 g --- yes
a r o z a u p a l a n a l a p u a z o r a --- yes
a r o z a u p a l a a l a p u a z o r a --- yes
a r o z a u p a l a l a p u a z o r a --- no
```

Время

	Array	Linked list
Add	0.00017	0.00099
Pop	0.00013	0.03658
Print	0.11078	0.18530

Память

См. вопрос 2.

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 ушёл, Last In – First Out (LIFO).

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

При реализации стека списком, память выделяется динамически по мере добавления новых элементов; число элементов в стеке ограничено только количеством доступной ОП.

При реализации стека массивом, выделяется фиксированный участок памяти; в стеке не может быть больше заданного числа элементов. Добавление нового элемента происходит путём смещения индекса вершины.

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

При реализации списком память из-под элемента освобождается после его удаления.

При реализации массивом память из-под элемента не освобождается, происходит лишь изменение значения индекса вершины.

4. Что происходит с элементами стека при его просмотре?

В общем случае доступ есть только к вершине стека; при просмотре она удаляется из стека, а указатель смещается далее. Для отображения состояния стека требуется последовательно проходить по всем его элементам, не «снимая» их.

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

Реализация стека массивом даёт огромный выигрыш во времени, поскольку не нужно каждый раз заново выделять и освобождать память. Тем не менее, в этом случае количество элементов в стеке жёстко ограничено – возможно либо переполнение стека, либо постоянный «излишек» памяти, отведённой под него. Способ реализации напрямую зависит от условий решаемой задачи – нужно знать примерное число элементов, которые могут храниться в стеке; можно ли пренебрегать переполнением; ограничен ли объём памяти.

Вывод

По результатам сравнений, использование массива при реализации стека даёт выигрыш в 6 раз при добавлении элементов и в 281 раз при их извлечении из стека. Разница во времени обуславливается тем, что при реализации списком происходит выделение и очистка блоков

памяти, в то время как при реализации массивом происходит только изменение значения элемента массива, и инкремент/декремент индекса вершины.

Реализация стека массивом даёт значительный выигрыш во времени, однако при этом максимальное количество элементов в стеке ограничено.

В моей реализации программы из-за слишком пёстрого интерфейса происходит фрагментация памяти при использовании стека на основе односвязного списка.