Отчёт

Лабораторная работа 4 по предмету «Типы и структуры данных».
Керимов Ахмед, ИУ7-34Б, Вариант 9.

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: Ввести арифметическое выражение типа: `число|знак| ... число|знак|число`, вычислить значение выражения. Реализовать возможность отображения текущего состояния стека.

Аварийные ситуации

1. Некорректный ввод. Будет выведено сообщение об ошибке.
2. Попытка добавления в заполненный массив. Будет выведено сообщение об ошибке.

Структуры данных

```
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;
};
```

```
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_;
};



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

Преобразование инфиксной формы в постфиксную

0. Глубина 0.
1. Читаем инфиксную форму слева направо.
2. Если прочитан операнд, откладываем его в очередь
3. Иначе, если это оператор:
    1. если приоритет оператора больше приоритета оператора на стеке, кладём его на стек.

2. Иначе достать все операторы со стека, приоритет которых больше или равен текущему, положить их в очередь.
4. Иначе, если это скобка:
    1. открывающаяся — инкрементировать переменную глубины,
    2. закрывающая — декрементировать переменную глубины. Если она
        1. отрицательная — кидаем ошибку
        2. положительная — повторяем 1–4.
5. Повторяем 1–4 пока не прочитаем всю инфиксную форму.
6. Достаём из стека все оставшиеся операторы, кладём их в очередь.

Рассчитать выражение в постфиксной форме

1. Операнд — кладём на стек.
2. Оператор — достаём два последних операнда со стека, выполняем операцию, результат кладём на стек.

Тесты

```
3 + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3    /* RC mandated: OK */
123                              /* OK */
3+4 * 2 / ( 1 - 5 ) ^ 2 ^ 3.14   /* OK */
(((((((1+2+3^(4 + 5)))))))       /* bad parens */
1^(2 + 3/4 * .1e5)!              /* unknown op */
(1^2)^3                          /* OK */
```

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.