МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
высшего образования
«Московский технологический университет»

МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИППО)

Отчет по практической работе № 1
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИСБОп-01-14 Карих Д.С.

Принял Шмелёва Д. В.

Москва 2017

Создание очереди, основанной на двух стеках

Очередь – структура данных, которая работает по принципу FIFO (First-In, First-Out), то есть все элементы, добавленные в неё, будут получены в том порядке, в котором они были добавлены.

Стек – структура данных, которая работает по принципу LIFO (Last-In, First-Out), то есть последний добавленный элемент будет получен первым.

Из этих двух определений следует, что очередь можно реализовать на двух стеках. При этом один стек будет использоваться в роли основного, а второй – в роли обратного.

В ходе данной практической работы класс Queue был реализован в качестве потомка класса Stack с добавлением обратного стека и переопределением метода push().

Реализация элемента (структура Node)

Элемент, который используется для хранения нужных нам данных в структурах, реализован по стандартному принципу.

```
struct Node {
    int data; // поле данных
    Node *next; // указатель на соседний элемент
};
```

Поле data имеет тип int, так как конкретный тип данных не был указан в задании. Поле next используется для хранения указателя на следующий элемент для придания связности нашей структуре данных.

Реализация стека (класс Stack)

Для создания стека нам понадобится всего одно поле данных: указатель на вершину стека.

```
Node *top; // указатель на вершину стека
```

Инициализируем указатель в конструкторе класса.

```
Stack() {
    top = NULL;
}
```

Далее реализуем метод push(int) для добавления новых элементов в стек. Метод выполняет три основных операции: сохраняет указатель на предыдущую вершину стека, заменяет вершину стека на новый элемент и добавляет в этот элемент указатель на предыдущую вершину стека.

```
void push(int a) {
    Node *top_old = top;
    top = new Node;
    top->data = a;
    top->next = top_old;
}
```

Создадим ещё два базовых метода: drop() для удаления текущего элемента и pick() для считывания значения из текущего элемента без его удаления.

```
void drop() {
    if (!isEmpty()) {
        Node *top_old = top;
        top = top->next;
        delete top_old;
    }
}

int pick() {
    if (isEmpty()) {
        throw "Stack is empty!";
    }
    return top->data;
}
```

Можно заметить, что метод pick() выбрасывает исключение, если в стеке отсутствуют элементы. Это позволяет избежать ошибок с лишними операциями чтения за счёт прерывания выполнения программы или отдельной процедуры.

Чтобы закончить реализацию стека, также создадим метод pop(), предназначенный для получения значения из текущей вершины стека. Этот метод также использует методы drop() и pick(), реализованные ранее.

```
int pop() {
    int data = pick();
    drop();
    return data;
}
```

За счёт того, что метод pick() выбрасывает исключение, метод pop() также будет выбрасывать исключение при попытке чтения из пустого стека.

Многие методы используют дополнительный метод isEmpty(), который позволяет узнать, остались ли в стеке элементы. Этот метод удобно применять в циклах при считывании элементов. Реализация его очень проста:

```
bool isEmpty() {
    return (top == NULL);
}
```

Наконец, реализуем деструктор, который должен будет очистить память, выделенную под элементы. Благодаря реализации метода drop(), деструктор можно реализовать в виде цикла, считывающего и уничтожающего элементы.

```
~Stack() {
    while (!isEmpty()) {
        drop();
    }
}
```

Реализация очереди на основе двух стеков (класс StackedQueue)

В связи со схожестью очереди со стеком, в нашем случае очень удобно применить принцип наследственности и сделать класс StackedQueue дочерним по отношению к классу Stack. Стоит заметить, что схожесть данных структур определяется не их принципом действия, а интерфейсом, который они предоставляют (методы push(), pop() и т. д.).

```
class StackedQueue : public Stack {
    . . .
}
```

Для начала, нам нужно добавить ещё одно поле типа Stack. Так как мы заранее знаем, что Stack будет только один (обратный), и заменять его в процессе работы не потребуется, объявим не указатель, а именно поле типа Stack в статичном виде.

```
Stack backward; // обратный стек
```

При этом инициализация и очистка данного поля не требуется, так как она уже предусмотрена во время компиляции. Данное поле будет неразрывно связано с каждым экземпляром класса StackedQueue.

Значительное преимущество объектно-ориентированного подхода с использованием наследования заключается в том, что нам осталось реализовать только метод push() для добавления элементов в очередь.

Из-за неоптимальности реализации очереди на двух стеках, метод push() выполняет значительно больше операций, чем метод pop(). То есть добавление элементов в очередь будет намного дороже в плане процессорного времени, чем аналогичное добавление элементов в стек.

Принцип действия метода push(): перемещаем элементы из основного стека в обратный, добавляем новый элемент в основной стек и, наконец, перемещаем элементы обратно в основной стек.

```
void push(int a) {
    while (!isEmpty()) {
        backward.push(pop());
    }

    Stack::push(a); // добавляем новый элемент

    while (!backward.isEmpty()) {
        Stack::push(backward.pop());
    }
}
```

Конструкция Stack:: используется для обращения к методу push() из родительского класса Stack, так как этот метод переопределён в дочернем классе StackedQueue.

Заключение

Реализация очереди на двух стеках возможна, однако требует гораздо больше операций при чтении или добавлении элементов (в зависимости от реализации). Гораздо более эффективной является реализация очереди с хранением указателей на начальный и конечный элемент.

Список источников

1. Википедия: Очередь (дата обращения: 21.02.17) // https://ru.wikipedia.org/wiki/Очередь_(программирование)
2. Википедия: Стек (дата обращения: 21.02.17) // https://ru.wikipedia.org/wiki/Стек
3. Википедия: Наследование (дата обращения: 21.02.17) // https://ru.wikipedia.org/wiki/Наследование_(программирование)