МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Оставь надежды всяк сие читающий (Карин Т.а.)

**Отчет по учебной практике**

**«**Структура хранения нескольких стеков в общей памяти**»**

**Выполнил:**

студент группы 381706­1

Карин Тимофей Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Проверил:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc1346524)

[3. Руководство пользователя 5](#_Toc1346525)

[4. Руководство программиста 6](#_Toc1346526)

[4.1. Описание структуры программы 6](#_Toc1346527)

[4.2 Описание структур данных 6](#_Toc1346528)

[4.3 Описание алгоритмов 7](#_Toc1346529)

[5. Заключение 10](#_Toc1346530)

[6. Список литературы 11](#_Toc1346531)

1. Введение

В одной из предыдущих лабораторных работ была подробно описана такая структура данных как стек. Следующим шагом в изучении структур данных является мультистек. Из названия легко понять, что это набор стеков. Но что же в нём особенного? Всё дело в способе хранения. В обычный стек на массиве может поместиться ограниченное число элементов. Допустим, что нам нужен не один стек, а несколько. В один из моментов может так случиться, что один из наших стеков полностью заполнен, в то время как в других места ещё есть. Но в этой ситуации сделать уже ничего нельзя. Конечно, можно было это предвидеть и сделать стек большего объёма, чтобы в него смогли поместиться все элементы, но это не рационально, так как наша драгоценная память будет расходоваться впустую.

В ситуации, когда имеется несколько стеков удобнее пользоваться такой структурой данных как мультистек. Он, как и обычный набор стеков, имеет ограниченный размер, но пока количество элементов в мультистеке меньше, чем его размер, мы можем положить элемент в любой из стеков. То есть размеры стеков, из которых состоит мультистек, имеют динамический размер, который зависит от количества элементов в этом и других стеках. Поэтому, мультистек удобен тем, что у нас не возникнет ситуации, когда один стек переполнен, а в остальных ещё есть место. Чем мультистек неудобен, так это реализацией методов. Но об этом в главе 4.3.

2. Постановка задачи

Основной целью данной работы является создание библиотеки для работы со структурой данных, хранящей несколько стеков (мультистека). Для этого необходимо:

* Разработать класс, для работы со стеками, но не выделяющий для них память;
* Создать класс, имеющий одну общую память, в которой будут храниться стеки; реализовать все методы для работы с этими стеками;
* Написать программу, демонстрирующую основные функции и возможности написанной библиотеки;
* Проверить работоспособность библиотеки с помощью тестов.

3. Руководство пользователя

Библиотека не предназначена для работы пользователей. Но этот недостаток удалось исправить. В файле *MStack.cpp* модуля *MStack*  хранится пример использования библиотеки. В нём пользователь может создать свой мультистек и управлять им. Рассмотрим основные моменты при работе с мультистеком.

Прежде всего программа попросит ввести размер создаваемого мультистека и количество стеков в нём. Важно ввести данные в соответствующем порядке. Вводить можно как через пробел, так и через кнопку ввода.

Затем пользователь увидит меню действий. Оно состоит из следующих пунктов:

1. Положить элемент в стек;
2. Взять элемент из стека;
3. Проверить стек на полноту;
4. Проверить стек на пустоту;
5. Выход.

Выбрав пункт 1) – 4) пользователю также будет необходимо ввести номер стека, с которым он хочет произвести действие. При выборе пункта 5) программа завершает свою работу.

4. Руководство программиста

4.1. Описание структуры программы

Программа состоит из 3 основных модулей:

* *MStackLib* – библиотека для работы с мультистеком. Содержит файлы *MStack.h* и NewStack.h.
* *MStack* – пример использования библиотеки. Подробно описан в главе 3.
* *MStackTest* – google тесты для библиотеки.

4.2 Описание структур данных

Рассмотрим класс *TNewStack*. Наследуется от класса *TStack* со спецификатором доступа public. Является шаблонным, *T –* шаблонный тип данных, присваиваемый элементам стека. Методы класса имеют спецификатор доступа public. Рассмотрим их:

* *TNewStack(int n, T\* m) –* конструктор. *n –* размер стека, m – указатель на массив элементов (при этом память не выделяется);
* *TNewStack(TNewStack<T> &A) –* конструктор копирования;
* *int Free() –* количество свободной памяти. Возвращает разницу между размером стека и количеством элементов в нём;
* *int GetSize() –* возвращает размер стека (поле *size*);
* *int GetTop() –* возвращает номер «верхнего» элемента (поле *top*);
* *void SetMas(int n, T\* m) –* присваивает указателю на массив указатель *m.*

Рассмотрим класс *TMStack*. Класс шаблонный, *T – ­*шаблонный класс, присваиваемый элементам стеков. Рассмотри поля и методы класса:

Со спецификатором доступа private:

* *int size* – общая длина;
* *T\* mas* – память для всего мультистека;
* int n – количество стеков в мультистеке;
* *TNewStack<T>\*\* h* – «псевдостеки» без выделения памяти;

Со спецификатором доступа public:

* TMStack(int \_n, int \_size) – конструктор, *n –* количество стеков, *size* – размер всего мультистека;
* TMStack(TMStack<T> &A) – конструктор копирования;
* int GetFreeMem() – возвращает количество свободной памяти (количество свободных элементов) во всём мультистеке;
* void Repack(int k) – функция «перепаковки». Предназначена для изменения размеров стеков. Подробно описана в главе 4.3;
* void Put(int n1, T p) – кладёт элемент *p* в стек *n1*;
* T Get(int n1) – возвращает элемент из стека *n1*;
* bool IsFull(int n1) – проверка стека *n1* на полноту;
* bool IsEmpty(int n1) – проверка стека *n1* на пустоту;

4.3 Описание алгоритмов

Создание мультистека

Прежде всего проверяем введённые параметры: если количество стеков или их размер меньше 1, то выбрасываем исключение. Если всё в порядке, то присваиваем соответствующим полям мультистека соответствующие параметры.

Затем, создаём массив объектов класса *TNewStack* (будем называть эти объекты псевдостеками или просто стеками)размера равного количеству стеков. Память под них не выделяется, поэтому также создаём массив элементов мультистека размера равного количеству элементов в мультистеке стеке.

Затем нам надо разделить память между нашими псевдостеками. Для этого создадим вспомогательный массив размеров псевдостеков. Разделять память будем следующим образом: делим нацело всю имеющуюся память на количество стеков и присвоим получившееся значение каждому из стеков. Остаток добавим к первому (нулевому) стеку.

Теперь надо указать псевдостекам, где хранятся их элементы. Для этого первому (нулевому) стеку указываем указатель на общую память (*mas*), а каждому последующему будем указывать этот указатель, сдвинутый на суммарное количество элементов предыдущих ему стеков. Мультистек создан, осталось только удалить вспомогательный массив размеров псевдостеков.

Копирование мультистека

Копируем размер (*size*), указатель на общую память (*mas*), создаём массив псевдостеков размера равного количеству стеков, копируем все элементы общей памяти.

Создаём вспомогательный массив размеров псевдостеков. Размер псевдостеков получаем с помощью метода (*GetSize()*).

Теперь надо указать, где начинается каждый из псевдостеков. Это делается так же, как и при создании стека. Первому присваиваем указатель на начало общей памяти, а все последующие сдвигаем на суммарное количество элементов в предыдущих стеках. Остаётся только удалить вспомогательный массив размеров псевдостеков.

Перепаковка мультистека (изменение размеров мультистека)

Для начала узнаём, есть ли у нас свободное место. Если вся память занята, выбрасываем соответствующее исключение.

Если память есть, то распределяем её между стеками. Делать это будем следующим образом: делим нацело всю свободную память поровну между стеками, а остаток добавляем тому, для которого вызывали перепаковку (переполненный, в который хотят добавить элемент).

Создаём два вспомогательных стека, в которых будут храниться указатели на старые начала псевдостеков (*oldStart*) и на новые (*newStart*). Ещё создаём стек, в котором будут храниться новые размеры псевдостеков (*newSize*).

В массив старых указателей (*oldStart*) записываем указатели до изменений, то есть первому присваиваем указатель на общую память, а последующие сдвигаем на суммарное количество элементов в предыдущих псевдостеках (свободные тоже считаются).

В массиве новых размеров (*newSize*) каждому стеку присваиваем количество элементов, которое в нём хранится и добавляем количество свободной памяти, которое выделили для этого стека.

С массивом новых указателей (*newStart*) поступаем так же, как и с массивом старых указателей (*oldStart*), но теперь количество элементов в псевдостеках изменилось, его надо брать из массива новых размеров (*newSize*). Таким образом, первому стеку ставится указатель на начало общей память, а каждый последующий указатель сдвигаем на суммарный размер всех предыдущих стеков, где размер берётся из массива *newSize.*

Теперь надо бы подряд перекопировать элементы, так как у нас есть все необходимые данные: новые и старые размеры и указатели на элементы. Но проблема в том, что мы не можем выделять новую память для элементов (почему? Неизвестно). А если мы будем копировать всё подряд, может возникнуть одна неприятная ситуация. Допустим на место старого элемента мы поставили другой новый. И теперь мы дошли до того момента, когда нужно копировать тот старый элемент. Где же его взять?! Уже нигде. Чтобы подобной ситуации не происходило, необходимо следовать следующему правилу: Если старое начало меньше нового начала стека или они равны, то копируем элементы в прямом порядке. Если новое начало меньше старого, то при копировании в прямом порядке, есть риск затереть нужные элементы. Поэтому, в этом случае, мы переходим к следующему стеку до тех пор, пока наше условие не будет нарушено. Элементы всех стеков, в которых выполняется данное условие, копируем в обратном порядке. Так мы не сможет удалить элемент, который ещё не был скопирован.

Теперь нужно присвоить нужные данные получившимся псевдостекам, а именно размер и указатель на первый элемент (то есть *newSize* и *newStart* соответственно).

Осталось удалить вспомогательные массивы ~~и поставить крест на этом методе.~~

5. Заключение

В ходе проделанной работы удалось получить следующие результаты:

* Разработана библиотека, предназначенная для работы с мультистеком;
* Написан пример использования этой библиотеки;
* Обработаны основные исключительные ситуации;
* Написаны тесты для большинства методов (с помощью фреймворка Google Test);

6. Список литературы

1. Ахо Альфред В, Хопкрофт Джон Э и Ульман Джеффри Д Структуры данных и алгоритмы [Книга]. - [б.м.] : Вильямс, 2003.
2. Лафоре Роберт Структуры данных и алгоритмы в Java [Книга]. - СПб : Питер, 2013. - 2 : стр. 704.
3. Павловская Т. А. C/C++ Программирование на языке высокого уровня [Книга]. - СПб : Питер, 2003.
4. Страуструп Бьерн Язык программирования C++ Бином, 2004.