ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 8

«Реализация собственной структуры данных. Куча»

Выполнила работу

Таволжанская Полина

Академическая группа C3100

Ментор: Вершинин Владислав Константинович

Санкт-Петербург

2024

**1. Введение**

Цель данной лабораторной работы заключается в углубленном изучении структуры данных "куча" (heap). В ходе выполнения этой работы особое внимание будет уделено основным методам работы с кучей, таким как создание, сложение, вычитание, умножение и деление. Задача заключается в реализации структуры данных "куча" с использованием динамических массивов. Это позволит не только применить теоретические знания на практике, но и глубже понять принципы работы с динамической памятью и ручного управления ею

**2. Теоретическая часть**

Для реализации данной лабораторной работы потребовалось:

1. Знание о динамических массивах и указателях в C++
2. Навыки ручного управления памятью
3. Теоретические знания о структуре данных «куча» и её свойствах
4. Понимание работы классов и методов в C++

**3. Реализация**

В ходе работы была реализована структура данных "куча" (heap) с использованием динамических массивов. Класс Heap был разработан с учетом основных операций, таких как создание, сложение, вычитание, умножение и деление.

Класс включает в себя следующие ключевые элементы:

1. Создание класса: конструктор класса Heap принимает размер кучи в качестве параметра и выделяет память для хранения элементов с помощью оператора new. Если выделение памяти не удалось, выбрасывается исключение std::runtime\_error, что позволяет избежать непредвиденных ошибок во время выполнения программы.
2. Приоритетность: в классе была добавлена логика приоритетности элементов не только по величине значения, но и по четности. Это реализовано в методе PriorityIsHigh, который определяет, какое число имеет более высокий приоритет: нечётные числа имеют более высокий приоритет, чем чётные. Это позволяет адаптировать кучу для специфических задач, где важна не только величина, но и четность чисел.
3. Методы для работы с кучей: метод **insert** добавляет новый элемент в кучу и восстанавливает свойства кучи с помощью механизма "всплытия". При добавлении нового элемента происходит сравнение его приоритета с приоритетом родительского элемента. Метод **extractMax** удаляет элемент с наивысшим приоритетом (корень) и восстанавливает свойства кучи через механизм "оседания".
4. Управление памятью: в классе реализован метод releaseMemory, который освобождает выделенную память вручную. Это позволяет избежать утечек памяти, что является важным аспектом при работе с динамическими массивами. Эффективное управление памятью обеспечивает стабильность работы программы и предотвращает потенциальные ошибки.
5. Сложность операций: временная сложность операций вставки (insert) и извлечения максимального значения (extractMax) составляет O(log n), где n — количество элементов в куче. Это связано с тем, что в худшем случае необходимо пройти от корня до листа кучи. Сложность операций сложения, вычитания, умножения и деления двух куч также составляет O(n\*log n), так как для каждого элемента требуется выполнение операции вставки в новую кучу.

**Ограничения**

Если не будет вызван метод «ReleaseMemory» для освобождения выделенной памяти, это приведет к утечкам памяти. Важно всегда освобождать память, чтобы избежать её переполнения при длительном использовании программы. Также если попытаться вставить элемент в полную кучу, будет выброшено исключение «overflow\_error». Помимо этих ограничений присутствуют такие, как отсутствие возможности хранения данных помимо целых чисел, отсутствие возможности изменять максимальный размер кучи динамически (что не должно быть проблематичным в связи с поставленными задачами, при которых размер кучи неизменяем).

**Приложение А**

**Листинг кода с решением поставленной задачи**

#include <iostream>

#include <stdexcept>

class Heap **{**

public**:**

int**\*** data**;** // указатель на динамический массив для хранения элементов кучи

int size**;** // макс кол-во элементов

int count**;** // текущее кол-во элементов

// конструктор определяет размер кучи + выделяет память под массив

Heap**(**int s**)** **:** size**(**s**),** count**(**0**)** **{**

data **=** **new** int**[**s**];** // выделение памяти для массива size s с помощью new

**if** **(!**data**)** **{** // проверка на успешное выделение памяти

**throw** std**::**runtime\_error**(**"Memory allocation failed"**);**

**}**

**}**

// определение приоритета элемента: нечётные числа имеют более высокий

// приоритет, чем чётные

bool PriorityIsHigh**(**int child**,** int parent**)** const **{**

**if** **(**data**[**child**]** **%** 2 **!=** 0 **&&** data**[**parent**]** **%** 2 **==** 0**)** **{**

**return** **true;** // нечётное число выше чётного

**}**

**if** **(**data**[**child**]** **%** 2 **!=** 0 **&&** data**[**parent**]** **%** 2 **!=** 0**)** **{**

**return** data**[**child**]** **>** data**[**parent**];**

**}**

**return** **false;**

**}**

// ПРОВЕРКА РАЗМЕРА

void checkSize**(**const Heap**&** other**)** const **{**

**if** **(**size **!=** other**.**size**)** **{**

**throw** std**::**invalid\_argument**(**"Size mismatch"**);**

**}**

**}**

// INSERTION

void insert**(**int value**)** **{**

**if** **(**count **>=** size**)** **{** // проверка на переполнение кучи

**throw** std**::**overflow\_error**(**"Heap is full"**);** // чтобы не выходить за память

**}**

data**[**count**++]** **=** value**;**

// но добавление по свойствам кучи

**for** **(**int index **=** count **-** 1**;** index **>** 0**;)** **{**

int parentIndex **=** **(**index **-** 1**)** **/** 2**;**

**if** **(**PriorityIsHigh**(**index**,** parentIndex**))** **{**

std**::**swap**(**data**[**index**],** data**[**parentIndex**]);**

index **=** parentIndex**;**

**}** **else** **{**

**break;**

**}**

**}**

**}**

// метод для получения и удаления элемента с наивысшим приоритетом

int extractMax**()** **{**

**if** **(**count **==** 0**)** **throw** std**::**underflow\_error**(**"Heap is empty"**);**

// чтобы не пытаться удалить из пустой кучи

int maxValue **=** data**[**0**];** // корень

data**[**0**]** **=** data**[--**count**];**

// придаём свойства кучи

**for** **(**int index **=** 0**;** index **\*** 2 **+** 1 **<** count**;)** **{**

int childIndex **=** index **\*** 2 **+** 1**;** // левый узел

// выбор правого узла, если он существует и имеет более высокий приоритет

**if** **(**childIndex **+** 1 **<** count **&&** PriorityIsHigh**(**childIndex **+** 1**,** childIndex**))** **{**

childIndex**++;** // переход к правому узлу

**}**

**if** **(**PriorityIsHigh**(**childIndex**,** index**))** **{** // проверка приоритета между

// текущим и нижним узлами

std**::**swap**(**data**[**index**],** data**[**childIndex**]);**

index **=** childIndex**;** // переход к нижнему узлу

**}** **else** **{**

**break;**

**}**

**}**

**return** maxValue**;**

**}**

// вывод "кучи"

void print**()** const **{**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

std**::**cout **<<** data**[**i**]** **<<** " "**;**

**}**

**}**

// УПР. ПАМЯТЬЮ (после обработки)

void releaseMemory**()** **{**

**delete[]** data**;** // освобождение памяти

data **=** **nullptr;** // обнуление указателя

**}**

// возвращает указатель на массив данных, который хранит элементы кучи

int**\*** getData**()** const **{**

**return** data**;**

**}**

// СЛОЖЕНИЕ

Heap **operator+(**const Heap**&** other**)** const **{**

checkSize**(**other**);** // проверка на совпадение размерности

Heap result**(**size**);** // future результат (далее то же самое для ост. методов)

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **+** other**.**data**[**i**]);**

**}**

**return** result**;**

**}**

// ВЫЧИТАНИЕ

Heap **operator-(**const Heap**&** other**)** const **{**

checkSize**(**other**);**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **-** other**.**data**[**i**]);**

**}**

**return** result**;**

**}**

// УМНОЖЕНИЕ

Heap **operator\*(**const Heap**&** other**)** const **{**

checkSize**(**other**);**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **\*** other**.**data**[**i**]);**

**}**

**return** result**;**

**}**

// ДЕЛЕНИЕ

Heap **operator/(**const Heap**&** other**)** const **{**

checkSize**(**other**);**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

**if** **(**other**.**data**[**i**]** **==** 0**)** **throw** std**::**domain\_error**(**"Division by zero"**);**

result**.**insert**(**data**[**i**]** **/** other**.**data**[**i**]);**

**}**

**return** result**;**

**}**

//--------------------------------------------------------------------------------

// ОПЕРАЦИИ СО СКАЛЯРАМИ, сохраняем св-ва используя уже написанный insert

// СЛОЖЕНИЕ

Heap **operator+(**int scalar**)** const **{**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **+** scalar**);**

**}**

**return** result**;**

**}**

// ВЫЧИТАНИЕ

Heap **operator-(**int scalar**)** const **{**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **-** scalar**);**

**}**

**return** result**;**

**}**

// УМНОЖЕНИЕ

Heap **operator\*(**int scalar**)** const **{**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **\*** scalar**);**

**}**

**return** result**;**

**}**

// ДЕЛЕНИЕ

Heap **operator/(**int scalar**)** const **{**

**if** **(**scalar **==** 0**)** **throw** std**::**domain\_error**(**"Division by zero"**);**

Heap result**(**size**);**

**for** **(**int i **=** 0**;** i **<** count**;** **++**i**)** **{**

result**.**insert**(**data**[**i**]** **/** scalar**);**

**}**

**return** result**;**

**}**

**};**