Оглавление

[Тема 1](#_Toc24619334)

[Преамбула 1](#_Toc24619335)

[Задачи 2](#_Toc24619336)

[Отчет 4](#_Toc24619337)

# Тема

Знакомство с контейнерами STL. Разработка класса двоичной кучи. Уход от рекурсии.

# Преамбула

Двоичная куча – простая и эффективная структура данных. Ее легко реализовать, используя контейнеры STL. Может служить хорошим упражнением для развития навыков программирования.

В этом задании предстоит разработать класс BinaryHeap, который реализует двоичную кучу на основе контейнера <vector> и поискать возможность оптимизации.

Алгоритмы кучи изложены в Лекции 5.

**Вызов конструктора члена класса с параметром**. В Задании 5 мы обнаружили, что к моменту выполнения кода конструктора, все члены класса уже созданы (их конструкторы исполнены). Это может быть причиной неэффективности. Рассмотрим пример.

class A {

vector<int> v;

public:

A(int n){

v = vector<int>(100);

}

};

В тексте программы, вызывается конструктор класса A, с указанием размера контейнера.

A a(100);

Согласно обнаруженного нами эффекта, сначала будет создан пустой вектор v, а в конструкторе мы заменим его на вектор нужного нам размера.

При этом, будет создан vector<int>(100), а затем присвоен нашему полю v, т.е. разрушится пустой v и будет создана копия временного вектора справа, а временный вектор будет разрушен по выходу из конструктора. Очень неэффективно!

Нужно сделать так, чтобы поле v создавалось сразу нужного размера!

Это достигается таким образом

A(int n)**: v(n)**{

}

Т.е. в конструкторе, после его параметров ставится двоеточие и указываются конструкторы полей класса, которые необходимо применять до перехода к выполнению бзоаого конструктора! Поля будут созданы сразу с нужными параметрами.

**Измерение времени**.

Для точного измерения времени в C++ программе пользуются следующим приемом. Включим заголовок <Windows.h>. Объявляем перед измеряемым участком 3 переменные:

LARGE\_INTEGER li1, li2, lif;

Сразу за объявлением вызываются 2 метода, которые включат секундомер:

::QueryPerformanceFrequency(&lif);

::QueryPerformanceCounter(&li1);

……

Сразу за измеряемым участком вызывается

::QueryPerformanceCounter(&li2);

Теперь время в микросекундах можно вычислить следующим образом:

int mcSec = (int)((double)((li2.QuadPart-li1.QuadPart)\*1000000)/(double)lif.QuadPart);

Эти приемы вам понадобятся при выполнении задания.

# Задачи

1. Начините новый консольный проект, назовите его BHRec. Подключите к нему потоки ввода-вывода. Убедитесь, что проект собирается и выполняется. Если необходимо, устраните проблемы.
2. Добавьте в проект класс BinaryHeap. Его можно сделать inline, т.е. реализовать все методы в h-файле. BinaryHeap.cpp можно удалить. Не забудьте при этом реализовать хотя бы пустой конструктор, который объявлен в h. Деструктор в этом классе не нужен.  
   Убедитесь, что проект собирается и выполняется. Если необходимо, устраните проблемы.
3. Добавьте в BinaryHeap.h модули STL – vector и exception, объявите в классе скрытое поле vector<int> heap.  
   Убедитесь, что проект собирается и выполняется. Если необходимо, устраните проблемы.
4. Добавьте в класс конструктор инициализации, который инициализирует кучу массивом целых чисел.
   1. Вспомните, как в C++ передается массив, как узнают его длину;
   2. Прочтите в преамбуле, как вызвать конструктор члена класса (heap) с параметром, и вызовите его так, чтобы не пришлось перестраивать массив.
   3. Выведите в конструкторе размер вектора heap на консоль.
5. В main подготовьте массив v целых на 10 элементов и заполните его какими-нибудь значениями, создайте объект BinaryHeap bh(v, 10).

Убедитесь, что проект собирается и выполняется. Если необходимо, устраните проблемы.

Убедитесь, по сообщению на консоли, что поле heap имеет размер 10.

1. В конструкторе инициализации заполните поле heap элементами массива-параметра. Удалите ненужный пустой конструктор по умолчанию.
2. Добавьте в класс вспомогательные операторы-оснастку.   
   - Оператор преобразования в целое, который просто возвращает размер поля heap   
   - и оператор []  
    int operator [] (int i)

Который:

* 1. Проверяет, что индекс лежит в допустимы пределах, если нет, то вбрасывает исключение out\_of\_range(“Index out of range”);
  2. Возвращает значение i-того элемента поля heap.

1. В main проверьте содержимое объекта, выводя в цикле все элементы на консоль через пробел. Вспомните, что сам объект может быть преобразован к int и сообщит размер массива.

Убедитесь, что проект собирается и выполняется. Если необходимо, устраните проблемы.

*Теперь, когда техническая часть закончена, можно приступать к реализации кучи.*

*Пользователю желательно всегда иметь правильную кучу и не заботиться о ее исправлении и пр.*

1. Добавьте в класс скрытый метод, fixHeap, по алгоритму из лекции. Учтите, что в лекции индексация массива начинается с 1, а heap индексируется с 0.
2. Добавьте в класс скрытый метод, buidHeap, по алгоритму из лекции. Учтите, что в лекции индексация массива начинается с 1, а heap индексируется с 0.
3. Вызовите в конструкторе инициализации после заполнения поля heap метод buildHeap, а ненужный вывод размера heap удалите, если он еще сохранился.

*Теперь наш класс действительно реализует кучу!*

Проверьте, что это так, выполнив программу и исследовав консольный вывод.

Попробуйте оптимизировать реализацию кучи. Для этого нам придется измерять время.

1. Объявите 2 целочисленных переменных, M и N, которые будут характеризовать параметры нашей задачи. Пусть M означает количество созданных в цикле объектов (итераций цикла), а N – их размер.   
     
   Их удобно сделать параметрами, т.е. значения M придется подобрать, чтобы время в микросекундах лежало в интервале от 6 до 30 mcsec для всех используемых N.

Можно начать с M=5 и N = 200.

1. Создайте массив целых v на N элементов и заполните его случайными числами rand()%N.
2. Объявите целочисленную переменную k, изначально равную 0. Это будет технический прием, который позволит нам “обмануть оптимизацию компилятора”.
3. Включите “секундомер”, как описано в Преамбуле.
4. Поместите цикл M раз, в котором
   1. создаются временные объекты BinaryHeap и инициализируются массивом v с правильным указанием его длины – N.
   2. первый элемент полученной кучи добавляется в k
5. Выключите секундомер.
6. На консоль выводится N и время в mcsec, а в следующей строке значение k (это нужно, чтобы компилятор не посчитал, что цикл пустой и его можно не выполнять!).
7. Убедитесь, что проект выполняется.
8. Задайте моду компиляции ***Release*** и подберите параметр M так, чтобы время исполнения для N=30 было в интервале 6-10 mcsec.

*Теперь, когда у нас есть “секундомер”, можно исследовать варианты оптимизации.*

В нашей реализации основной метод fixHeap реализован рекурсивно. Рекурсия не только “съедает” память стека, но и приводит к некоторой неэффективности. Интересно, насколько это существенно? Попробуем избавиться от рекурсии в этом методе.

Удобнее всего скопировать файл BinaryHeap.h, назвать его BinaryHeapOpt.h и аналогично переименовать класс, определенный в файле и его конструктор, а затем включить полученный файл в проект (Добавить => Существующий элемент …).

1. Затем:
   1. Замените рекурсию в методе fixHeap на цикл while;
   2. Постарайтесь максимально оптимизировать цикл;
2. Замените в main создание объектов BinaryHeap на BinaryHeapOpt и оцените полученный эффект по времени исполнения.   
   Сделайте выводы. Что можно сказать о вычислительной нагрузке рекурсии?

# Отчет

Подготовьте краткий отчет с описанием вашего исследования.

Отчет и очищенный архив проекта поместите в качестве ответа на задание.