Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине

«Языки программирования»

Работу выполнил
студент группы СКБ-201 _____ Г.П. Кашкин
подпись, дата

Работу проверил _____ С.А. Булгаков
подпись, дата

Москва 2021

Содержание

Постановка задачи

Разработать программу на языке Си++ (ISO/IEC 14882:2020), демонстрирующую решение поставленной задачи.

Общая часть

Разработать набор классов, объекты которых реализуют типы данных, указанные ниже. Для классов разработать необходимые конструкторы, деструктор, конструктор копирования, а также методы, обеспечивающие изменение отдельных составных частей объекта. Используя перегрузку операторов (operator) разработать стандартную арифметику объектов, включающую арифметические действия над объектами и стандартными типами (целыми, вещественными, строками – в зависимости от вида объектов), присваивание, ввод и вывод в стандартные потоки (используя операторы «<<» и «>>»), приведение к/от базового типа данных. Организовать операции в виде конвейера значений, с результатом (новым объектом) и сохранением значений входных операндов.

Задачи

1. Дата и время, представленные целочисленными переменными: год, месяц, день, час, минута, секунда. Базовый тип: `uint64_t` формат представления unix time. Реализовать возможность преобразования в/из формата представления filetime (целое 64-х разрядное значение, представляющее число интервалов по 100 наносекунд, прошедших с первого января 1601 года).

2. Целое произвольной длины (во внешней форме представления в виде строки символов-цифр). Базовый тип: `std::string`.

3. Год «от Адама»,имеющий внутреннее представление в виде целочисленных переменных: ин дикт, круг солнцу, круг луне. Диапазоны значений (циклические): индикт 1—15, круг солнцу 1—28, круг луне 1—19. Ежегодно каждая переменная увеличивается на 1. Итоговое значение вычисляется как произведение переменных(диапазона на некоторый множитель; переменные независимы), а хранимое значение является остатком от деления (на диапазон), при этом 0 соответствует максимум. Необходима возможность отображения/задания как в виде одного числа, так и виде трех. Реализовать возможность преобразования в/из формата представления «от рождества Христова» используя соответствие 1652 = 7160 «от Адама».

4. Разреженная матрица, представленная динамическим массивом структур, содержащих описания ненулевых коэффициентов: индексы местоположения коэффициента в матрице (целые) и значение коэффициента (вещественное).

1 Алгоритм решения задачи

1.1 Задание 1

Для решения задачи использовались функции стандартной библиотеки и базовые арифметические операции. Для синхронизации двух форм представления использовались функции `std::mktime` и `std::gmtime`, для преобразования в человекочитаемую строчку - `std::ctime`. Для конвертации в формат представления `Filetime` использовалась константа разницы эпох microsoft и unix.

1.2 Задание 3

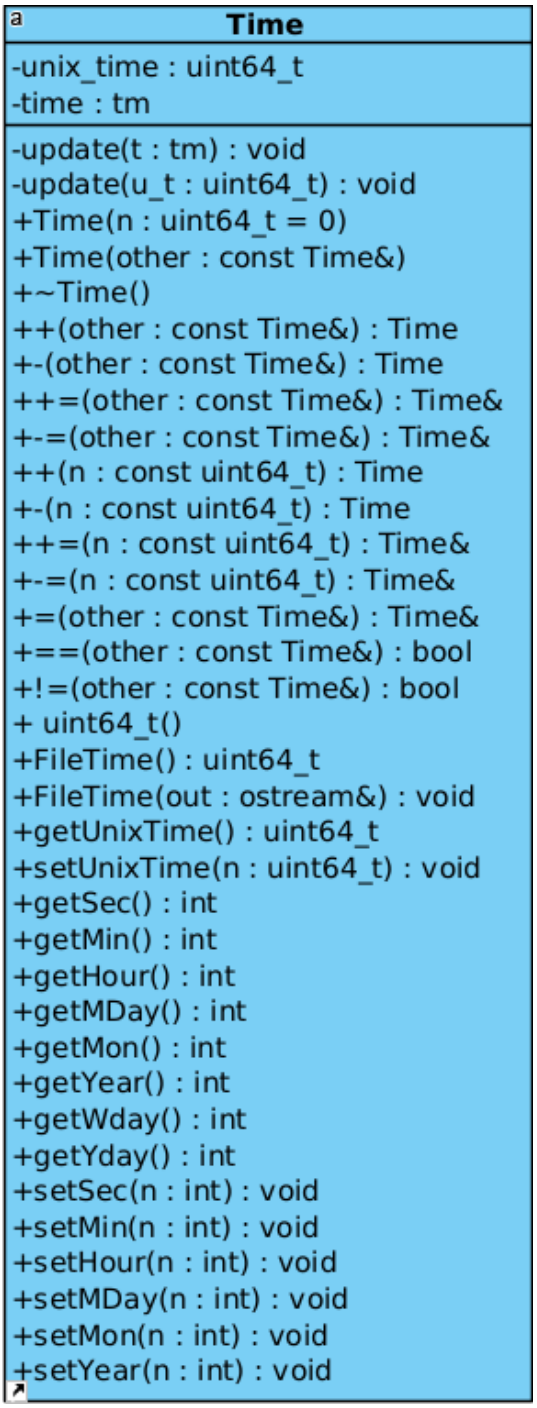
Для решения задачи было необходимо разработать двустороннюю конверцию из тройки параметров в номер года. Преобразование в тройку параметров происходит через многократное деление с остатком на границы значений. В обратную сторону - происходит перебор всех возможных годов, которых всего $15 \cdot 28 \cdot 19 = 7890$, то есть алгоритм фактически имеет константную асимптотику. Для преобразования в года от Рождества Христова необходимо вычесть константу данную в условии (5508).

1.3 Задание 4

Для оптимального решения задачи используется декартово дерево. Классическая реализация последнего базируется на двух операциях - `merge` и `split`, которые объединяют и делят деревья соответственно. Алгоритмы рекурсивны и базируются на поддержании инвариантов дерева, то есть для каждой ноды верно следующее: ключ у всех элементов левого поддерева строго меньше ключа ноды, а у всех элементов правового - больше (помимо этого поддерживается аналогичный инвариант по приоритетам, но я не буду их рассматривать, так как они случайны и необходимы только для лучшей средней полноты дерева). Проверка инварианта работает через отключаемый флагом макрос и делается повсюду. Таким образом, вставка и удаление элементов так же имплементированы через `split` и `merge`. Поиск по ключу реализован как спуск от корня дерева, поэтому для него необходимы только индексы потомков. Напротив для итераторов требуется нахождение следующей и предыдущей по ключу вершины, что задает необходимость в хранении предков ноды. Хотя увеличение итератора формально и имеет асимптотику $O(\log n)$, но усредненно работает за $O(1)$, поэтому может классифицироваться как bidirectional. Для запланированной разработки `random_access_iterator` уже заложен подсчет размера поддерева для каждой ноды, при помощи него получится за $O(\log n)$ перемещаться на произвольное число нод.

2 Выполнение задания

2.1 Задание 1



Для решения данной задачи разработан класс `Time` хранящий в себе одновременно структуру `std::tm` и переменную `uint64_t unix_time`, содержащие совокупную информацию о дате и времени в целочисленном формате и количество секунд с начала unix эры соответственно. При модификации любого из полей запускается функция `update`, которая обновляет синхронизирует второе поле. Синхронизация происходит посредством функций `std::mktime` и `std::gmtime` конвертирующими один формат значений в другой. Перегрузка `operator<<` основана на функции `std::ctime` возвращающей красивую строку с информацией о дате и времени по указателю на `unix_time`. Помимо этого в классе реализована базовая арифметика, explicit конверция в `uint64_t`, функции доступа (с префиксами get и set), конвертация в формат `Filetime`. Так как все поля тривиальны - используется деструктор по умолчанию, для конструкторов используются списки инициализации. Реализованы copy и move семантики.

2.2 Задание 3

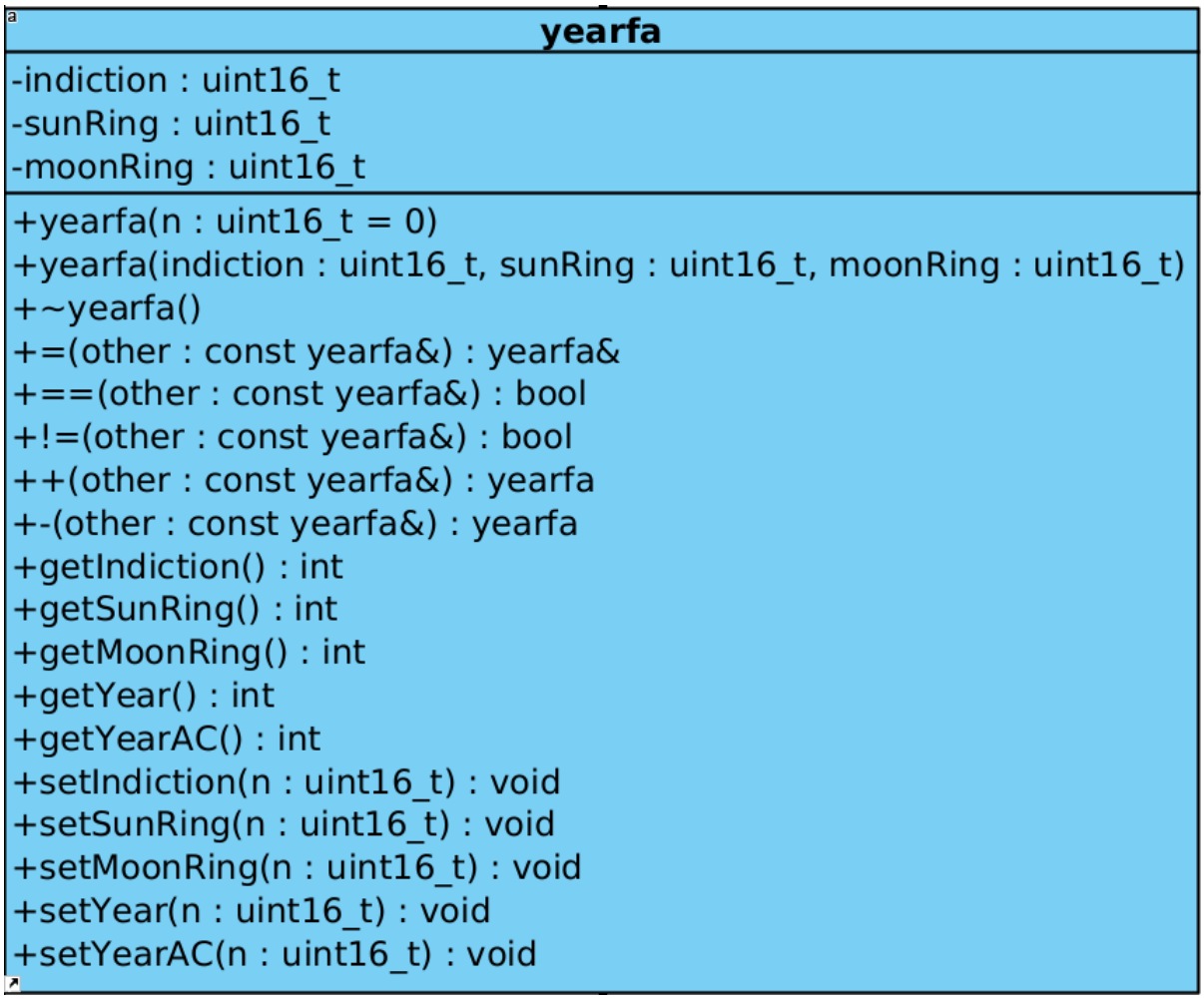


Рис. 1: UML 2.0 diagram for yearfa class

Для решения данной задачи разработан класс `yearFA`, описывающий календарь «от Адама». Концепция самого календаря базовая, поэтому все функции, кроме рассчитывающих год по параметрам прописаны в `inline` формате внутри `yearfa.hpp`. Последние реализованы посредством перебора, данный календарь поддерживает только 7890 лет, следовательно асимптотика алгоритма - $O(1)$. Так как все поля тривиальны - используется деструктор по умолчанию, для конструкторов используются списки инициализации, копирование так же тривиально и реализуется компилятором.

2.3 Задание 4

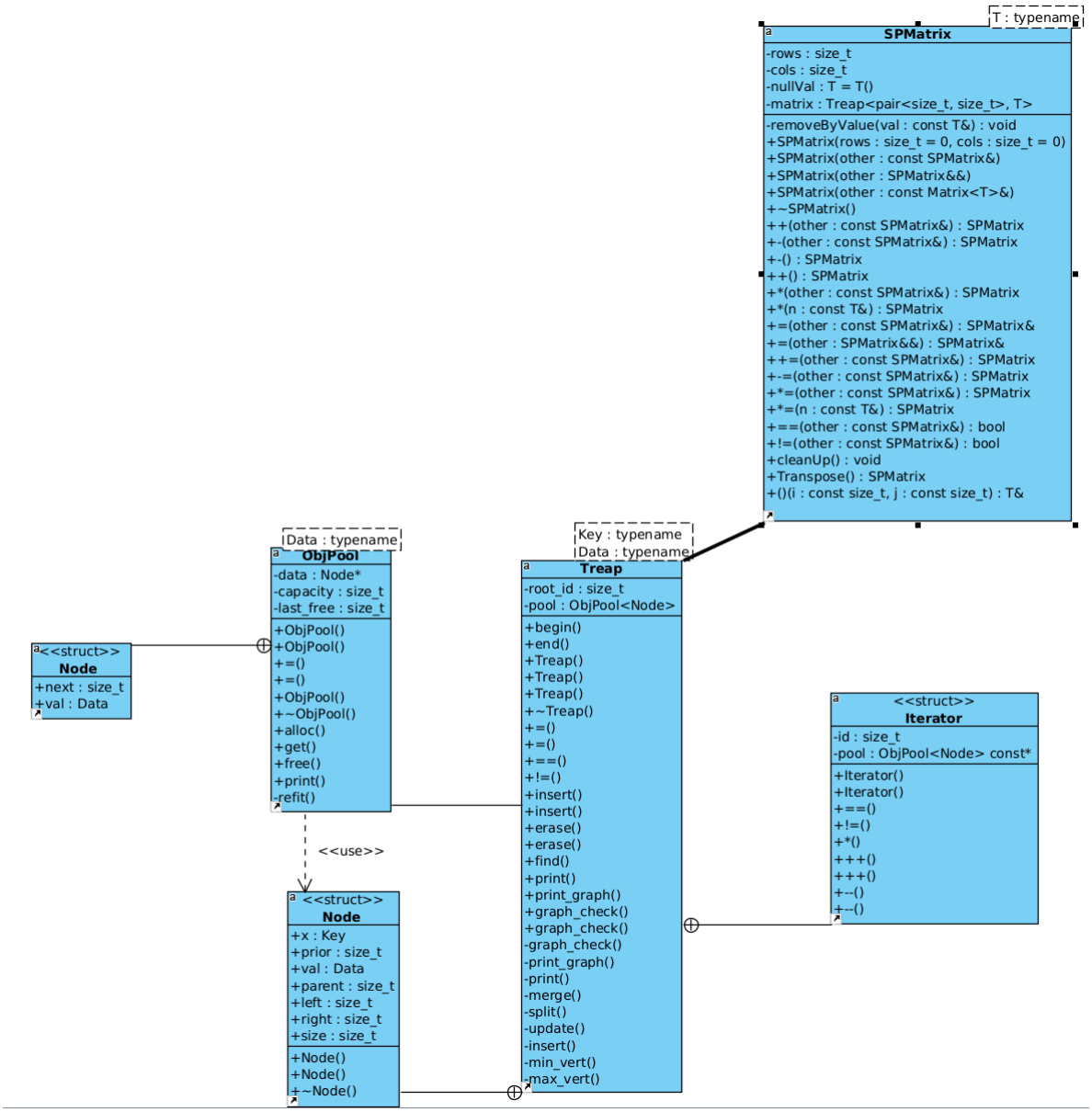


Рис. 2: UML 2.0 diagram for spmatrix class

Для решения данной задачи разработан класс `spmatrix`, представляющий разреженную матрицу (sparse matrix), он базируется на классе `treap` - декартовом дереве, позволяющем усредненно за $O(\log(n))$ вставку, поиск и удаление элемента, построение дерева происходит за $O(n)$. Для `treap` определена внутренняя структура ноды, хранящая необходимые сервисные значения, то есть индекс предков и детей, случайный приоритет, размер поддерева данной вершины, ключ и полезную нагрузку. Для оптимизации выделения и освобождения памяти используется класс `ObjPool` (object pool), представляющий из себя связанный список доступных для выделения нод `treap`. При аллоцировании занимается крайняя в списке нода, если в `ObjPool` заканчиваются свободные ноды, он переаллоцируется целиком на память в два раза больше. Так как при переаллоцировании ломаются все указатели на выделенные ноды, приходится использовать индексы. На декартовом дереве реализована поддержка `bidirectional_access_iterator`, функции вставки, поиска и удаления элементов по ключу. Помимо этого есть несколько сервисных функций облегчающих отладку и небольшой макрос того же свойства. Хранение размеров поддеревьев в нодах `treap` реализовано для расширения итераторов в `random_access`, но по-

следнее пока не готово. Разыменование итератора возвращает пару из ключа и его полезной нагрузки. `treap` как и `spmatrix` базируется на механике шаблонов, таким образом декартово дерево можно использовать в других проектах, а матрицы поддерживают произвольные типы данных полезной нагрузки. Доступ к объектам матрицы реализован через перегрузки `operator()`, для чтения может выдаваться сервисный нулевой объект, для модификации ищется или аллоцируется нода по ключу. Кроме базовых арифметических операций реализовано также умножение, работающее за $O(n \cdot m)$, где n - число ненулевых элементов первой матрицы, а m - число столбцов второй. Для всех классов реализована copy и move семантика.

3 Получение исполняемых модулей

Для всего проекта использовалась система сборки `cmake`. В конфигурации системы сборки прописаны три режима компиляции: basic, sanitizer и debug с разными уровнями придирчивости (использование посредством флага `-DCMAKE_BUILD_TYPE=*`), в каждом из них прописано использование требуемого стандарта `c++14` и компилятора `clang`. Помимо этого `cmake` автоматически скачивает с github и подключает библиотеку для unit тестов `GoogleTest`, которая используется для проверки корректности программ. Для каждого задания создан отдельный конфиг, который рекурсивно подключается в корневом `CMakeLists.txt`.

4 Тестирование

Тестирование производится при помощи библиотеки `GoogleTest`, везде, где позволяет логика используется многократный проход теста с генератором случайных значений.

4.1 Задание 1

4.1.1 Операторы

Операторы тестируются на повторяемость результатов операций, корректность копирования и стрессоустойчивость всех методов.

4.1.2 Функции доступа

Функции доступа многократно тестируются на стрессоустойчивость на случайных значениях, в том числе выходящих за рамки ожидаемых.

4.2 Задание 3

4.2.1 Операторы

Немногочисленные операторы тестируются на на повторяемость результатов операций, корректность копирования и стрессоустойчивость.

4.2.2 Фунции доступа

Функции доступа проверяются на стабильность и соответствию инварианта, указанного в условии.

4.3 Задание 4

4.3.1 Treap

Класс декартового дерева как и ObjPool тестировался вручную при помощи красивых функций вывода состояния и проверки встроенной инвариантов на крайних значениях. Всего три различных теста для Treap и один для ObjPool.

4.3.2 Простое умножение

Группа тестов проверяющих корректность операции умножения. Состоит из базового теста на корректность умножения на единичную матрицу, предподсчитаного умножения матриц небольших размеров.

4.3.3 Умножение случайных

Группа тестов проверяющих корректность операции умножения на случайных матрицах. Заполнение матриц происходит при помощи класса Matrix разработанного ранее и стабильного. С ним же и сравниваются результаты операций.

4.3.4 Сложение случайных

Как и в прошлой группе проверяется корректность сложения случайных матриц произвольного размера, результат сравнивается с заведомо корректным результатом Matrix.

4.3.5 Транспонирование

Проверка корректности операции транспонирования разреженной матрицы.

4.3.6 Копирование и перемещение

Многократное копирование и перемещение случайно заполненных матриц и проверка операций присваивания.

Приложение A

A.1 Файл `test.cpp`

Прикладывание нескольких тысяч строк кода в pdf это тлен...

Приложение Б

Б.1 Файл `test.cpp`