Лабораторная работа №1 по курсу «Проектирование программного обеспечения интеллектуальных систем» (2 курс 1 семестр)

Тема: Основы объектно-ориентированного программирования на языке C++

- понятие класса и объекта;
- поля и методы класса;
- статические методы класса;
- определение методов внутри и вне класса;
- принцип инкапсуляции;
- управление доступом (области видимости);
- конструктор и деструктор;
- указатель `this`;
- ссылка на объект;
- использование спецификаторы `const` в возвращаемом типе, в типе аргументов и после метода класса;
- конструктор по умолчанию;
- конструктор копирования;
- спецификатор `mutable` для поля класса;
- динамическое выделение памяти под объекты (операторы `new` и `delete`);
- ссылка на себя;
- перегрузка операторов +, −, *, /, =, ==, !=, >, >=. <, <= и т.д.;
- перегрузка потоковых операторов (<<, >>);
- операторы-члены и не-члены;
- `std::string` (строки в стиле C++).
- документирование кода при помощи `doxygen`:
  - краткое и полное описание;
  - тэг;
  - тэги `file`, `author`, `brief`, `details`, `param`, `return`, `see`;
  - генерация документации.

1 Задание

Реализовать на языке C++ один из нижеперечисленных вариантов и написать и сгенерировать документацию при помощи doxygen. Для возможности тестирования классов написать тестовую программу с меню или набор unit-тестов. В случае написания unit-тестов необходимо проверить не менее 30 тестов случаев с использованием библиотеки UnitTest++.

Каждый из реализованных классов должен иметь следующие свойства:

- инкапсуляция;
- отделение консольного пользовательского интерфейса программы от реализации класса;
- конструктор копирования и оператор =, если это уместно;
- деструктор, если это необходимо;
- операторы для сравнения (операторы ==, !=);
- операторы для работы с потоками ввода (>>) и вывода (<<);
- взаимная независимость класса и пользовательского интерфейса, использующего его;
- отделение объявления класса в h-файл, а реализации — в cpp-файл.

1.1 Вектор

Описать класс для работы с вектором, который задается координатами концов в трехмерном пространстве. Класс должен реализовывать следующие возможности:

- получение координат концов вектора;
- вычисление длины вектора;
- сложение двух векторов (операторы +, +=);
- вычитание двух векторов (операторы −, −=);
- произведение двух векторов (операторы *, *=);
- произведение вектора на число (операторы *, *=);
- деление двух векторов (оператор /, /=);
- определение косинуса между двумя векторами (оператор ^);
- операторы для сравнения двух векторов (операторы >, >=, <, <=);

1.2 Англо-русский словарь

Описать класс, реализующий англо-русский словарь в виде бинарного дерева (в узлах хранится пара слов, причем английское слово является ключом). Класс должен реализовывать следующие возможности:

- добавление нового английского слова и перевода для него (оператор += для работы как со строками в стиле C - char *, так и в стиле C++ - std::string);

- удаление существующего английского слова из словаря (оператор −=);
- поиск перевода английского слова (оператор []);
- замена перевода английского слова (оператор []);
- определение количества слов в словаре;
- возможность загрузки словаря из файла;

1.3 Прямоугольник

Описать класс прямоугольника со сторонами, параллельными осям координат. Вершины прямоугольников имеют должны иметь целочисленные координаты. Класс должен реализовывать следующие возможности:

- получение координат вершин;
- перемещение;
- изменение размера;
- увеличение размера на единицу по каждой из осей (оператор пре- и постинкремента ++);
- уменьшение размера на единицу по каждой из осей (оператор пре- и постдекремента −−);
- построение наименьшего прямоугольника, содержащего два заданных прямоугольника (оператор +);
- построение наименьшего прямоугольника, содержащего два заданных прямоугольника, с присваиванием (оператор +=);
- построение прямоугольника, являющегося общей частью (пересечением) двух прямоугольников (оператор −);
- построение прямоугольника, являющегося общей частью (пересечением) двух прямоугольников, с присванием (оператор −=);

1.4 Теория множеств

Описать класс "Множество". Класс должен реализовывать следующие возможности:

- элементом множества может быть другое множество;
- проверка на пустое множество;
- добавление элемента;
- удаление элемента;
- определение мощности множества;
- проверка принадлежности элемента множеству (оператор []);
- объединение двух множеств (операторы + и +=);
- пересечение двух множеств (операторы ∗ и ∗=);
- разность двух множеств (операторы − и −=);
- построение булеана данного множества;

1.4.1 Множество

Описать класс "Неориентированное канторовское множество". Класс должен реализовывать следующие возможности:

- формирование множества из строки, как из `char *`, так и из `std::string`; например:

  `{a, b, c, {a, b}, {}, {a, {c}}}.`

1.4.2 Мультимножество

Описать класс "Неориентированное мультимножество". Класс должен реализовывать следующие возможности:

- формирование множества из строки, как из `char *`, так и из `std::string`; например:

  `{a, a, c, {a, b, b}, {}, {a, {c, c}}}).`

1.5 Игры

Все программы-игры должны иметь консольный интерфейс и меню для игры.

1.5.1 Крестики-нолики

Описать класс, реализующую игру "Крестики-нолики"(поле произвольного размера) между двумя игроками. Класс должен реализовывать следующие возможности:

- проверки возможности установки крестика/нолика в указанной позиции;
- получение значения указанной позиции (оператор []);
- установка значения указанной позиции (оператор []);
- проверка победы одного из игроков;

1.5.2 Пятнашки

Описать класс, реализующий игру-головоломку "Пятнашки". Начальное размещение номеров — случайное. Реализовать методы для осуществления перестановки клеток, для проверки правильной расстановки клеток. Класс должен реализовывать следующие возможности:

- случайное начальное размещение номеров;
- перестановка клеток;
- получения значения клетки (оператор `[]`);
- проверка правильной расстановки клеток.

1.5.3 Кубик Рубика

Описать класс, реализующий игру-головоломку "Кубик Рубика". Класс должен реализовывать следующие возможности:

- случайное начальное размещение цветов;
- загрузка начального размещения цветов из файла;
- поворот грани кубика;
- проверка правильной расстановки цветных клеток;

1.6 Математика

1.6.1 Натуральная дробь со знаком

Описать класс, реализующий тип данных "Натуральная дробь со знаком". Натуральная дробь всегда должна хранится в сокращенном виде. Класс должен реализовывать следующие возможности:

- получение определителя, знаменателя и целой части;
- сложение двух натуральных дробей (операторы +, +=);
- сложение натуральной дроби с целым (операторы +, +=);
- вычитание двух натуральных дробей (операторы −, −=);
- вычитание из натуральной дроби с целого (операторы −, −=);
- произведение двух натуральных дробей (операторы ∗, ∗=);
- произведение натуральной дроби и целого (операторы ∗, ∗=);
- деление двух натуральных дробей (операторы /, /=);
- деление натуральной дроби на целое (операторы /, /=);
- операторы пре- и постинкремента, пре- и постдекремента (операторы ++, −−);
- сравнение двух натуральных дробей и натуральной дроби с целым (операторы >, >=, <, <=);
- приведение к `double`;

1.6.2 Многочлен от одной переменной

Описать класс многочлена от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Класс должен реализовывать следующие возможности:

- получение значений коэффициентов (оператор `[]`);
- вычисление значения многочлена для заданного аргумента (оператор `()`);
- сложение двух многочленов (операторы +, +=);
- вычитание двух многочленов (операторы −, −=);
- произведение двух многочленов (операторы ∗, ∗=);
- деление двух многочленов (операторы /, /=);

1.6.3 Длинное целое со знаком

Описать класс, реализующий тип данных "длинное целое со знаком"и работу с ними. Длина целого числа должна быть неограниченной. Класс должен реализовывать следующие возможности:

- оператор преобразования длинного целого к целому;
- сложение двух длинных целых (операторы +, +=)
- сложение длинного целого с целым (оператор +, +=);
- вычитание двух длинных целых (операторы −, −=)
- вычитание из длинного целого целого (оператор −, −=);
- произведение двух длинных целых (операторы ∗, ∗=)
- произведение длинного целого и целого (оператор ∗, ∗=);
- деление двух длинных целых (операторы /, /=)
- деление длинного целого на целое (оператор /, /=);
- пре- и постинкремент (++), пре- и постдекремент (−−);
- сравнение двух длинных целых (операторы >, >=, <, <=);
- сравнение длинного целого с целым (операторы >, >=, <, <=);

1.7 Теория матриц

Описать класс, реализующий тип данных "Вещественная матрица". Класс должен реализовывать следующие возможности:

- матрица произвольного размера с динамическим выделением памяти;
- пре- и постинкремент (++), пре- и постдекремент (−−);

1.7.1 Вещественная матрица

Класс должен реализовывать следующие дополнительные возможности:

- изменение числа строк и числа столбцов;
- загрузка матрицы из файла;
- извлечение подматрицы заданного размера;
- проверка типа матрицы (квадратная, диагональная, нулевая, единичная, симметрическая, верхняя треугольная, нижняя треугольная);
- транспонированные матрицы;

1.7.2 Вещественная матрица (усложненный)

Класс должен реализовывать следующие дополнительные возможности:

- сложение двух матриц (операторы +, +=);
- сложение матрицы с числом (операторы +, +=);
- вычитание двух матриц (операторы −, −=);
- вычитание из матрицы числа (операторы −, −=);
- произведение двух матриц (оператор ∗);
- произведение матрицы на число (операторы ∗, ∗=);
- деление матрицы на число (операторы /, /=);
- возведение матрицы в степень (оператор ^, ^=);
- вычисление детерминанта;
- вычисление нормы;

1.8 Абстрактные машины

Описать классы, реализующие абстрактную машину, ее ленту, программу и правила, каретку, алфавит, строку. Набор классов зависит от вида абстрактной машины. Количество классов может варьироваться, но минимально должно быть три класса. Классы должны реализовывать следующие возможности:

- загрузка программы из потока ввода;
- загрузка состояние ленты из потока;
- добавления/удаления/просмотр правил;
- задания/изменения значений на ленте;
- осуществление шага и интерпретации всей программы (можно при помощи опрераторов инкремента и декремента);

Написать программу, которая принимает в качестве аргумента командной строки путь к файлу, который содержит начальное абстрактной машины и программу для интепретации. После запуска программа считывает содержимое этого файла и осуществляет интерпретацию правил. Если в командной строке был указан аргумент `-log`, тогда после выполнения каждого правила на консоль должна выводится информация о состоянии абстрактной машины.

1.8.1 Машина Поста

Описать классы, реализующие машину Поста.

1.8.2 Машина Тьюринга

Описать классы, реализующие машину Тьюринга.

1.8.3 Нормальные алгорифмы Маркова

Описать классы, реализующие нормальные алгорифмы Маркова.

2 Литература

- Х. Дейтел "Как программировать на C++"5-ое издание 2008 года

  Для выполнения лабораторной работы необходимо прочитать главы 3, 9, 10, 11. Ссылочный тип и спецификатор const и mutable не описаны в этих главах. Эта книга рекомендуется.

- Б. Страуструп "Язык программирования C++"

  Для выполнения лабораторной работы необходимо прочитать главы 5 и 6. Ссылочный тип и спецификатор const описаны в более ранних главах. Для новичка эта книга сложновата.

- С. Макконнелл "Совершенный код"

  В этой книге необходимо прочитать главу 22, которая связана с тестированием ПО, которая проводится разработчиком.

- Doxygen на русском

3 Часто встречающиеся ошибки

Студенты забывают использовать ссылки C++ & для избежания копирования объектов. Например:

```
// Накладные расходы на копирование объекта
void f(std::string str);

// Передача по ссылке без копирования.
// В данном случае не забываем про const,
// чтобы нельзя было изменить объект из функцииvoid
f(const std::string &str);
```

Отсутствие спецификатора `const` в местах, где он необходим по смысле. Необходимо как можно чаще его использовать в возвращаемых типах, в типах аргументов и для задания константных методов.

Это делает код строже, позволяет лучше специфицировать интерфейс для внешнего пользователя и избежать ошибок (компилятор будет ругаться на попытки изменить `const`-объект).

Отсутствие понимания разницы между оператором + и += (с присваиванием) (тоже самое и для других похожих операторов). Внимательно изучите возвращаемые типы этих операторов и разберитесь с понятием "Ссылка на себя".

Студенты не могут ответить на вопрос "В каких случаях нужен конструктор копирования?".