Методические рекомендации к выполнению лабораторных работ по дисциплине «Информатика и Программирование» 2-го курса бакалавриата ИВТ

I модуль

Содержание

1 Классы и наследование

1.1 Классы

Сложной программе вовсе необязательно выглядеть сложно. И у объектно-ориентированных языков программирования есть несколько способов упрощения кода. В структурных языках программирования последовательность команд, связанных общей задачей, обычно объединяют в подпрограмму (т.е. функцию или процедуру). Где бы и сколько бы раз она ни понадобилась, теперь достаточно написать одно её имя, а не копировать одни и те же строки кода – чем не шаг к упрощению? Можно пойти ещё дальше, сгруппировав переменные и функции, для чего существует *пространство имён* (`namespace`). Однако это пространство только одно; как быть, если нужно несколько одинаковых по структуре наборов данных?

На помощь приходят классы. Один раз описав класс с его *полями* (переменными класса) и *методами* (функциями класса), можно создать любое количество объектов этого класса, обладающих такой же структурой, но хранящих свои значения. Однако на этом их удобство не заканчивается.

Класс позволяет задать специальные методы, которые объект этого класса самостоятельно вызовет в самом начале и самом конце своего существования. *Конструктор* чаще всего используется для инициализации полей объекта и, при необходимости, выделения памяти, а *деструктор* – для очистки выделенной памяти. Поскольку объект может быть создан разными способами, существует несколько разновидностей конструктора:

Конструктор по умолчанию не принимает аргументов	`Sample()`
Конструктор инициализации принимает от одного до нескольких аргументов и на их основании устанавливает значения полей	`Sample(int i, char c)`
Конструктор копирования принимает ссылку (обычно константную) на объект того же класса, и копирует значения его полей	`Sample(const Sample &other)`

Также классы позволяют установить доступность своих полей и методов:

```
class Sample {
   public:
      //доступ везде, где определён этот класс
   protected:
      //доступ только в методах этого класса или его наследниках
   private:
      //доступ только в методах этого класса
}
```

Для предоставления регулируемого доступа к недоступному полю, в классе могут быть использован т.н. *селектор*, возвращающий значение поля и *модификатор*, присваивающий полю новое значение (переданное в аргументе), если оно пройдёт некоторую проверку. В ряде объектно-ориентированных языков, селектор и модификатор являются обычными методами.

1.1.1 Структура класса

Класс должен включать в себя

- конструктор по умолчанию
- конструктор инициализации
- конструктор копирования
- поля класса, требуемые по условию задачи
- методы доступа (селекторы и модификаторы) для скрытых (private) полей класса

Если в классе окажутся открытые (public) поля, это должно быть обосновано.

В классе присутствуют перечисленные поля и методы *Класс не допускает хранение ошибочных значений*	**40%**

1.1.2 Тестирование

Чтобы убедиться в том, что методы класса работают как положено, их необходимо подвергнуть проверке. Для этого мы воспользуемся функцией `assert()` из библиотеки `assert.h`. Эта функция остановит выполнение программы и выдаст ошибку, если её аргумент окажется ложным (`false`) или равным нулю. Если несколько раз вызвать эту функцию с различными предположениями вида `ожидаемое значение == рассчитанное значение` в качестве аргумента, тогда завершение программы без ошибки будет означать, что во всех вызовах функции `assert()` аргументы были истинными (`true`), а все наши предположения оказались верны.

Тестированию должны подвергнуться следующие функции:

- конструктор по умолчанию: (поле == значение по умолчанию)
- конструктор инициализации: (поле == подходящее значение инициализации) или (поле == значение по умолчанию, если значение инициализации ошибочно)

- конструктор копирования: (поле копии == поле оригинала)
- методы доступа полей класса: (поле == подходящее новое значение) или (поле != ошибочное новое значение)

Перечисленные функции успешно проходят тестирование **30%**

1.1.3 Оформление кода

Хотя у всех могут быть свои предпочтения касательно размера отступов, расположения скобок и других мелочей, придерживаясь одного стиля оформления кода можно существенно упростить чтение, а иногда и понимание кода своей программы. Работая с библиотеками `stl` и `Qt` следует принять во внимание, что принятые ими стили немного различаются:

	`stl`	`Qt`
Имена переменных и функций	`snake_case`	`camelCase`
Имена классов	`snake_case`	`PascalCase`

Код программы выдержан в одном стиле
Есть логика в именах переменных и классов **15%**

1.1.4 Эффективность кода

Для проверки работоспособности исходного кода программы требуется преобразовать его в исполняемый машинный код. Этот нетривиальный процесс называется *компиляцией*, а выполняющие его программы, соответственно, называются *компиляторами*. Компиляторы различаются в своей способности упрощать и оптимизировать исходники, генерируя эффективный машинный код, однако они не всесильны и подсказки от программиста могут им здорово помочь.

Одной из таких подсказок может быть использование модификатора `const`. Поля и объекты, объявленные с таким модификатором не могут быть изменены в течение своего существования, а методы класса при вызове не изменяют значения полей.

`const int Class::method(int value)`	**возвращает неизменяемое значение**
`int Class::method(const int value)`	не изменяет значение **аргумента**
`int Class::method(int value) const`	не изменяет значения **полей класса**

Другой способ подразумевает использование модификатора `&`. В отличие от оператора `&`, который используется в *выражениях* и возвращает адрес переменной, этот модификатор используется в *объявлениях* полей, методов и аргументов, чтобы показать, что мы передаём его значение *по ссылке*, обращаясь напрямую к области памяти, в которой лежит этот параметр. Без использования модификатора `&` параметр передаётся *по значению*, т.е. копируется, а действия производятся над его независимой локальной копией.

`void Class::increase(int value){ value++; }`	Увеличивает **копию** переменной `value`
`void Class::increase(int &value){ value++; }`	Увеличивает переменную на единицу
`int Class::getValue(){ return this->value; }`	Возвращает **копию поля** `value`
`int& Class::getValue(){ return this->value; }`	Возвращает поле `value` (и потенциальную возможность его изменить)

Хотя копирование переменных базовых типов почти не стоит ни памяти, ни процессорного времени, на копировании больших объектов обычно можно здорово сэкономить. Правильное применение модификаторов `const` и `&` может повысить эффективность работы программы, а неправильное – привести к лишней трате ресурсов или дырам в безопасности.

`bool Data::isEqual(Data other) { ... }`	Неоправданное копирование!
	Решения:
`bool Data::isEqual(Data &other)`	работает с оригиналом объекта
`bool Data::isEqual(const Data &other)`	гарантирует неизменность оригинала
`int& Class::value() { return _value; }`	Возможность изменить приватное поле!
	Решения:
`int Class::value()`	возвращает копию поля
`const int& Class::value() const`	возвращает неизменное значение поля

Методы, не изменяющие объекты, объявлены как константные (const)
Передача по ссылке (&) позволяет избежать ненужного копирования **15%**

1.2 Коллекции

Даже при написании простой программы можно столкнуться с обработкой больших объёмов информации. Хотя для хранения однородных данных можно использовать статические массивы, при удалении и добавлении элементов хотелось бы избежать лишних операций копирования и выделения памяти. Специально для выполнения этой задачи и хранения объектов других классов, существуют так называемые классы-*контейнеры* или *коллекции*, каждый со своим способами добавления, удаления и расположения объектов в памяти. В рамках этой лабораторной работы потребуется написать собственную реализацию одного из этих классов:

Вариант	**Коллекция** (`stl` / `Qt`)	**Добавление**	**Удаление**	**Доступ** (чтение/запись)
1–3	Стек `std::stack` / `QStack`	в конец	с конца	с конца или по итератору
4–6	Очередь `std::queue` / `QQueue`	в конец	с начала	с начала или по итератору
7–9	Двусвязный список `std::list` / `QLinkedList`	в произвольном месте		с начала, с конца или по итератору
10–12	Одномерный массив `std::vector` / `QVector`	в произвольном месте		по индексу
13–15	Двухмерный массив – / –	–	–	по двум индексам

1.2.1 Структура коллекции

В классе-коллекции должны быть определены

- конструктор по умолчанию (варианты 1–12)
- конструктор инициализации (варианты 10–15, размер коллекции как параметр)
- конструктор копирования
- методы добавления и удаления элементов (варианты 1–12)
- класс элемента коллекции (итератор) (варианты 1–9)
  - объект класса из предыдущего задания
- метод доступа к элементу по итератору (варианты 1–9) или индексу (варианты 10–15)
- метод, удаляющий все хранимые объекты
- метод, возвращающий число хранящихся объектов
- прочие методы, если таковые требуются по условию задачи

Класс соответствует перечисленным пунктам
Нет ошибок при выделении и очистке памяти **30%**
Нет возможности нарушить работоспособность коллекции

1.2.2 Работа с файлами

В классе-коллекции также должны присутствовать два метода – для записи данных в произвольный файл и загрузки данных из произвольного файла. Можно выбрать один из способов представления данных:

Читаемый текст	`std::fstream` / `QTextStream`
Двоичные данные	`std::fstream (std::ios::binary)` / `QDataStream`
Другие форматы: XML, JSON и т.д.	`QXmlStreamWriter` + `QXmlStreamReader`; `QJsonDocument`

Содержимое коллекции записывается в произвольный файл и читается из файла без ошибок **15%**

1.2.3 Тестирование

Как и в предыдущей работе, тестирование производится при помощи функции `assert()`. Для тестирования потребуются две дополнительные функции:

① Функция проверки двух коллекций на равенство, поэлементно проверяющая их соответствующие объекты
(может быть членом класса-коллекции)

② Функция последовательного и пронумерованного вывода элементов коллекции на экран
(**не** может быть членом класса-коллекции)

Тестированию должны подвергнуться

- конструктор копирования: (копия == оригинал)①,
  затем изменение копии и проверка (копия != оригинал)
- конструктор инициализации (варианты 10–15):
  (размер коллекции == введённый параметр) для массива или
  (размер коллекции == произведение введённых параметров) для двумерного массива
- методы добавления и удаления объектов (варианты 1–12):
  (размер коллекции == размер коллекции перед добавлением одного объекта + 1) и
  (размер коллекции == размер коллекции перед удалением одного объекта - 1)
- метод, удаляющий все хранимые объекты:
  (размер непустой коллекции после удаления всех объектов == 0)
- метод доступа к элементу: вывод на экран всех хранимых объектов ②
- методы чтения и записи в файл:
  (непустая коллекция, сохранённая в новый файл == коллекция, прочитанная из того же файла)①

Перечисленные функции успешно проходят проверку **25%**

1.2.4 Оформление кода

Код программы выдержан в одном стиле
Есть логика в именах переменных и классов **15%**

1.2.5 Эффективность кода

Методы, не изменяющие объекты, объявлены как константные (const)
Передача по ссылке (&) позволяет избежать ненужного копирования **15%**

1.3 Наследование

1.3.1 Реализация наследования

Все общие для классов методы и поля объявлены в родительском классе, а уникальные – только в классах-наследниках *Имеется виртуальный метод, позволяющий отличить классы между собой* *Класс-коллекция хранит объекты обоих классов без потери функционала*	25%

1.3.2 Графический интерфейс

GUI даёт возможность работы с объектами обоих классов, предоставляет доступ ко всем функциям контейнера и даёт возможность выполнить поставленную задачу	20%

1.3.3 Тестирование

Программа не допускает ошибок и правильно решает поставленную задачу	25%

1.3.4 Оформление кода

Код программы выдержан в одном стиле *Есть логика в именах переменных и классов*	7%

1.3.5 Оформление интерфейса

Каждый из элементов интерфейса выполняет свою определённую задачу *Интерфейс предостерегает пользователя от ошибок и рационально использует пространство*	8%

1.3.6 Эффективность кода

Методы, не изменяющие объекты, объявлены как константные (const) *Передача по ссылке (&) позволяет избежать ненужного копирования*	7%

1.3.7 Эффективность интерфейса

Интерфейс автоматически адаптируется к изменениям *Интерактивные элементы интерфейса выключаются или скрываются в моменты, когда нет возможности или смысла с ними взаимодействовать*	8%

Варианты лабораторных работ

0 Текст задания.

1. Класс для первой лабораторной работы: его поля и специфичные методы
2. Контейнер для второй лабораторной работы (его структура данных): хранение объектов классов из п. 1, п. 3 и реализация своих специфичных методов
3. Класс для третьей лабораторной работы: его поля и методы, частично совпадающие с таковыми у класса из п. 1.

1 На одном из столов офиса растёт стопка макулатуры и одноразовой посуды. Сотрудники опасаются, что в определённый момент равновесие нарушится и стопка упадёт. Разработайте для них программу, позволяющую воссоздать стопку и определить, находится ли она в равновесии.

1. Лист бумаги: ширина, высота, толщина, координаты по двум осям
2. Стопка макулатуры (стек кусков бумаги): метод проверки на наличие равновесия
3. Бумажная тарелка: радиус, толщина, координаты по двум осям

2 Некоторых писателей отталкивает сложность современных текстовых редакторов. Разработайте для них программу с минимальным функционалом: хранение и редактирование текста с возможностью отмены и возвращения изменений.

1. Операция редактирования текста: тип операции (вставка или удаление), текст и позиция изменённой подстроки
2. История правок (2 стека операций над текстом): методы отмены и восстановления правок в заданном тексте
3. Операция замены текста: старый текст, новый текст и позиция изменённой подстроки

3 Бакалавры с факультета математики открыли для себя множество комплексных чисел и теперь нуждаются в новом калькуляторе. Калькулятор должен поддерживать простейшие арифметические операции и производить вычисления, записанные в стеке.

1. Комплексное число: действительная и мнимая часть, арифметические операции (сложение, вычитание, умножение и деление)
2. Калькулятор (стек комплексных чисел и арифметических операций): вычисление значения хранящегося выражения
3. Операция: знак операции ($+$, -, *, /, $\Re$, $\Im$, $=$), количество операндов

4 Престижная сеть ресторанов быстрого питания решила существенно сократить расходы, заменив часть персонала роботами. Для автоматизации процессов на кухне были закуплены программируемые роботы-повара. Для работы повара нужно представить процесс приготовления каждого блюда как очередь из ингредиентов и действий над ними. Разработайте программу для хранения и редактирования таких процессов.

1. Ингредиент: название, единица измерения и количество (в этих единицах)
2. Рецепт (очередь ингредиентов и действий над ними)
3. Операция над ингредиентом: действие (нарезать, упаковать, пожарить, положить и т.п.), длительность действия

5 Студенты с кафедры робототехники запускают радиоуправляемый автомобиль. На автомобиль ещё не успели поставить GPS-модуль, поэтому его местоположение приходится вычислять по переданным с пульта командам. Напишите программу, позволяющую определить координаты автомобиля на основании переданных с пульта команд.

1. Команда движения: текущее время, скорость
2. Последовательность команд (очередь команд движения и поворота): расчёт координат автомобиля в заданное время
3. Команда поворота: текущее время, радиус поворота

6 Филолог Филипп уже долгое время работает на кассе ресторана быстрого питания, однако с появлением автоматов самообслуживания он начал беспокоиться за своё рабочее место и искать способы получить одобрение начальства. Одним из таких способов является сбор пожертвований в различные фонды, например, фонд поддержки филологов. Чтобы похвастаться своей продуктивностью, он хочет составить подробный отчёт с ежедневной выручкой. Поскольку с математикой Филипп не в ладах, ему нужна программа, рассчитывающая выручку за произвольный период времени.

1. Покупка: время и общая сумма покупки
2. Кассовый аппарат (очередь покупок и пожертвований): расчёт выручки за заданный период времени
3. Пожертвование в фонд: время и сумма пожертвования, название фонда

7 Профессора кафедры прикладной математики любят заходить в студенческую столовую после долгого дня ведения лекций и написания научных статей. При выборе обеда каждый профессор уделяет особое внимание калорийности, поскольку он хочет восполнить дневные затраты на умственную деятельность. Напишите программу, позволяющие хранить блюда и их сочетания в порядке увеличения энергетической ценности и производить поиск ближайшего блюда или обеда по заданной калорийности.

1. Блюдо: название, вес, содержание на 100 грамм продукта жиров, белков, углеводов, органических кислот и пищевых волокон, метод расчёта энергетической ценности на основе этих данных

2. Меню (двусвязный список блюд и обедов): автоматическая сортировка блюд по калорийности (при добавлении), поиск ближайшего блюда по заданной калорийности
3. Комплексный обед: первое, второе и десерт (блюда, входящие в меню), метод расчёта энергетической ценности

8 Коммивояжёр в перерывах между разъездами тратит массу времени на сбор чемодана. Ему бы очень пригодился список вещей, отсортированный по их удельной ценности (цена/объём). Напишите программу, позволяющую автоматически составить такой список.
1. Предмет: название, объём, цена
2. Список вещей (двусвязный список предметов и монет): автоматическая сортировка предметов по удельной ценности (при добавлении)
3. Стопка монет: цена, функция расчёта объёма (минимальное число монет достоинством в 1, 5, 10, 50 и 100 у.е., каждая из которых занимает единичный объём)

9 Сотрудники института биоинформатики хотят восстановить первичную структуру белка, для чего требуется объединить его (частично пересекающиеся) фрагменты. Требуется написать программу, которая хранит белковые последовательности (в виде списков аминокислот) и умеет их объединять, если аминокислоты в конце одной последовательности совпадут с аминокислотами в начале другой.
1. Протеиногенная аминокислота: способ показывать (и принимать) своё полное название и однобуквенное обозначение
2. Белок (двусвязный список аминокислот): операция сложения, возвращающая результат сложения (с пересечением) двух белков
3. Нестандартная аминокислота: способ показывать и принимать своё полное название

10 Блогер Боря собирается публиковать плейлисты для изменения настроения. Для каждого музыкального произведения в своей коллекции он рассчитал настроение по шкалам грустное-весёлое и спокойное-интенсивное. Ему нужна программа, которая принимает начальное и конечное настроение и выдаёт список музыкальных композиций с минимальными скачками настроения (между композициями).
1. Музыкальная композиция: название, исполнитель, настроение
2. Сборник (массив музыкальных композиций и попурри): метод, принимающий начальное и конечное настроение и возвращающий новый список композиций, меняющий своё настроение от начального до конечного с минимальным средним размером скачка
3. Попурри: название, начальное и конечное настроение композиции

11 В новом автосалоне требуется расположить несколько автомобилей и автоподиумов, причём так, чтобы автомобили и автоподиумы не пересекались между собой, а на подиумах стояли только автомобили соответствующих марок. Напишите приложение, проверяющее выполнение этих условий.
1. Автомобиль: название марки, габариты, координаты, угол поворота
2. План автосалона (массив автомобилей и подиумов): габариты, координаты, метод проверки на наличие контактов между автомобилями и стенами автосалона
3. Автоподиум: название марки, радиус, координаты

12 Магистров с факультета математики застукали за распитием кофе в лаборатории. В связи с этим дежурного сотрудника обязали вести электронный журнал учёта кофе. В помощь лаборатории, напишите программу для работы с таким журналом, проверяющую правильность заполнения журнала.
1. Запись в журнале: время доступа к кофеварке, информация о сотруднике (Ф.И.О., должность и учёная степень) и оказанное влияние на объём кофе (больше нуля, если прибавилось и меньше, если убавилось)
2. Журнал (массив записей): ёмкость колбы, метод проверки журнала на наличие ошибок (кофе было меньше нуля или больше, чем вмещает колба)
3. Запись дежурного: время наблюдения, текущий объём кофе

13 До квадратного королевства, в котором каждый день была одна и та же погода, добрался первый циклон. Опасаясь возможной паники среди населения, король издал указ построить метеостанции для измерения погодных условий. В казне хватило денег, чтобы застроить всё королевство метеостанциями с шагом в одну королевскую милю, но часть средств загадочным образом пропала, потому вместо нескольких метеостанций пришлось поставить дешёвые флюгеры. Напишите программу, рассчитывающую погоду в произвольной точке, интерполируя данные с ближайших метеостанций и флюгеров.

1. Показания метеостанции: температура, давление, а также направление и скорость ветра
2. Метеоцентр (двумерный массив флюгеров и метеостанций): расчёт погодных показателей в произвольной точке королевства
3. Флюгер: направление и скорость ветра

14 У кафедры электротехники внезапно закончилась лицензия на всё программное обеспечение. Разработайте программу, позволяющую рассчитать напряжения в узлах заданной цепи.

1. Резистор: номер, сопротивление
2. Матрица сопротивлений (двумерный массив элементов электрической цепи): частота тока (f), список источников тока, расчёт потенциалов в узлах цепи методом узловых потенциалов
3. Индуктивность/ёмкость: номер, реактивное сопротивление, метод расчёта полного сопротивления (по частоте тока f)

15 Будущих бухгалтеров отталкивает сложность современных электронных таблиц. Разработайте для них программу с минимальным функционалом: хранение числовых и текстовых данных в таблице с изменяемым размером.

1. Ячейка электронной таблицы: числовое или текстовое значение, может быть пустой
2. Таблица (двумерный массив ячеек): методы вычисления суммы и среднего арифметического всех числовых данных в произвольном столбце
3. Ячейка с формулой: диапазон ячеек (адреса первой и последней ячейки) и операция над диапазоном (сумма, произведение, среднее значение), метод вывода результата операции (или ошибки, если расчёт невозможен)

A user interface is like a joke:
If you have to explain it, it's not that good