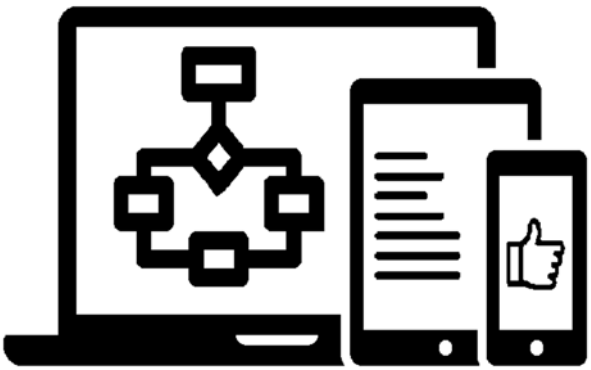
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические рекомендации к курсовому проектированию для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения



Могилев 2021

УДК 621.01
ББК 36.4
О87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой ««Программное обеспечение информационных технологий» «28» января 2021 г., протокол № 6

Составители: ст. преподаватель Ю. В. Вайнилович;
ст. преподаватель О. В. Сергиенко

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации предназначены к выполнению курсового проекта по дисциплине «Объектно-ориентированное программирование» для студентов направления подготовки 1-40 05 01 «Информационные системы и технологии (по направлениям)» очной и заочной форм обучения.

Учебно-методическое издание

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Ответственный за выпуск	В. В. Кутузов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью курсового проекта является закрепление основ и углубление знаний приемов программирования на языке C++, получение практических навыков на всех этапах создания программного продукта: от постановки задачи до практической реализации, сопровождающейся документацией и инструкциями по его использованию. При выполнении курсового проекта студентам рекомендуется обратить внимание на качество создаваемых программ и оформления документации.

На проверку преподавателю предоставляются оформленная и скрепленная пояснительная записка и программа на любом электронном носителе (исходные и исполняемый файлы).

Также можно предоставить для проверки материалы курсового проекта в электронном виде. Проверка курсовых проектов в электронном виде прекращается за две недели до начала экзаменационной сессии.

1 Общие требования к курсовому проекту

1.1 Тематика курсовых проектов

Тематика подразделяется на типовую и выбираемую студентом самостоятельно.

В типовом проекте предлагается разработать программное средство по обработке информации о некоторой предметной области, созданию и обработке файлов, используемых для долговременного хранения информации. При этом студенту необходимо самостоятельно определить структуру и характеристики значений вводимых данных, обосновать используемый способ обмена информацией, создать файл во внешней памяти, предусмотреть ряд функций по обработке информации.

Каждое задание на курсовое проектирование уточняется с преподавателем. Примерный перечень заданий на курсовое проектирование представлен в приложении А.

1.2 Общие требования к курсовому проекту

Среда программирования – Microsoft Visual Studio 2017 и выше.

Язык программирования – C++.

Структура проекта – многофайловая.

Парадигма программирования – объектно-ориентированная.

Способ хранения данных – файлы.

Разрабатываемый интерфейс должен быть понятным и защищенным от случайных ошибок.

Ввод исходных данных должен осуществляться либо с клавиатуры, либо из файла (по выбору пользователя).

Результаты работы программы должны сохраняться в текстовый файл по запросу пользователя.

1.3 Функциональные требования к курсовому проекту

Программа реализуется в виде консольного или Windows-приложения. Основные возможности, которые должна выполнять программа, представлены ниже.

1 Работа с файлом данных:
   а) создание файла;
   б) открытие существующего файла;
   в) удаление файла.
2 Работа с данными:
   а) режим редактирования:
      – просмотр выбранных данных;
      – добавление новых данных;
      – удаление данных;

– редактирование записи;

б) режим обработки данных:

– выполнение индивидуального задания;
– поиск данных (как минимум по трем различным параметрам);
– сортировка (как минимум по двум различным параметрам).

Предусмотреть:

– возможность навигации по пунктам меню;

– подтверждение выполнения необратимых действий, например, удаление файла должно подтверждаться сообщением вида «Вы действительно хотите удалить файл?»;

– обратную связь с пользователем, например, вывод сообщений об успешности создания файла / удаления записи / и т. д.

1.4 Требования к программной реализации курсового проекта

1 Программный код должен быть оформлен в соответствии с правилами, приведенными в приложении В.

2 Один метод решает только одну задачу (например, не допускается в одном методе считывать данные из файла и выводить их на консоль – это две разные функции). При этом внутри метода возможен вызов других методов.

3 Выполнение операций чтения / записи в файл должно быть сведено к минимуму (т.е. после однократной выгрузки данных из файла в массив/вектор дальнейшая работа ведется с этим массивом/вектором, а не происходит многократное считывание данных из файла в каждом методе).

4 Следует избегать длинных методов и глубокой вложенности: текст функции/метода должен умещаться на один экран, а вложенность блоков и операторов должна быть не более трех.

5 Выносите код логически независимых модулей в отдельные .cpp-файлы и подключайте их с помощью заголовочных .h-файлов.

2 Содержание курсового проекта

2.1 Содержание пояснительной записки

В состав пояснительной записки к курсовому проекту входит следующее.

Титульный лист.

Бланк задания курсового проекта.

Введение.

1 Описание предметной области.

1.1 Постановка задачи.

1.2 Объектно-ориентированный анализ предметной области.

2 Проектирование программного средства.

2.1 Проектирование структур данных.

2.2 Проектирование взаимодействия с пользователем.

2.3 Проектирование интерфейса.
3 Реализация программного средства.
3.1 Реализация классов и структур данных.
3.2 Реализация требований.
3.3 Реализация концепций объектно-ориентированного прогаммирования.
3.4 Описание структуры проекта.
3.5 Реализация интерфейса взаимодействия с пользователем.
4 Тестирование программного модуля.
Заключение.
Список использованных источников.
Приложение А. Текст программы.

Образец оформления титульного листа представлен в приложении Б.

2.2 Содержание графической части проекта

Графическая часть проекта выполняется на листах формата А3–А1, где изображается следующее:

- снимки экрана;
- структура меню;
- диаграмма классов.

3 Требования к содержанию разделов пояснительной записки

3.1 Введение

Приводится цель, задачи и ожидаемый результат курсового проекта.

3.2 Описание предметной области

3.2.1 Постановка задачи. Выполняют подробное описание решаемой задачи: описывают назначение разрабатываемой программы, перечисляют входные и выходные данные программы, формулируют требования к программе. В требованиях должно быть отражено следующее.

Требования к функциональным характеристикам программ. Это перечень функций, которые должна реализовывать программа.

Пример.

Разрабатываемая программа должна обеспечивать возможность выполнения перечисленных ниже функций:

- *ввода исходных данных;*
- *запуска процесса вычислений;*
- *просмотра результатов вычислений;*
- *записи результатов вычислений в файл.*

Требования к входным и выходным данным программы. Входными данными для программы являются данные, вводимые пользователем через её интерфейс. Выходные данные – это информация, получаемая в результате выполнения

программы. В записке обосновывают выбор способа ввода/вывода данных. Ввод данных может осуществляться с клавиатуры или из файла, а вывод – в файл или на монитор компьютера в виде текста, таблицы, графика.

Пример.

Требования к организации ввода входных данных:

– *левая граница отрезка*, *содержащего корень уравнения. Способ ввода – с клавиатуры или из файла;*

– *правая граница отрезка*, *содержащего корень уравнения. Способ ввода – с клавиатуры или из файла;*

– *погрешность вычисления. Способ ввода – с клавиатуры;*

– *предусмотреть проверку на принадлежность входных данных допустимым диапазонам и повторение ввода при ошибочных данных.*

Требования к организации вывода выходных данных:

– *значение функции в точке*, *являющейся корнем уравнения. Способ вывода – на монитор компьютера в виде строки текста и в файл;*

– *количество итераций при нахождении корня управления. Способ вывода – на монитор компьютера в виде строки текста и в файл;*

– *график функции. Способ вывода – на экран в виде таблицы и в файл для построения данной зависимости с помощью программного продукта Microsoft Excel.*

Требования к интерфейсу пользователя. Следует указать тип интерфейса пользователя.

Пример.

Тип пользовательского интерфейса – консольное приложение.

Пример.

Разрабатываемая программа должна иметь графический интерфейс пользователя в виде Windows-формы.

Требования к обработке ошибок. Перечисляются возможные ошибки пользователя при работе с программой, указываются способы диагностики и защиты от этих ошибок.

Пример.

При выполнении программы необходимо предусмотреть обработку следующих ошибок:

– *неправильный формат исходных данных;*

– *невозможность выделения отрезка*, *содержащего корень уравнения*;

– *ошибка чтения / записи в файл.*

В результате выполнения данного должны быть выделены все требования к проектируемому программному средству.

3.2.2 Объектно-ориентированный анализ предметной области. Словесно описывается указанная в задании предметная область, выделяются сущность предметной области. Описываются характеристики объектов предметной области. Термины «класс», «поле» в данном пункте не употребляются.

Пример.

Охотничий магазин продает оружие и амуницию. Все товары в магазине характеризуются ценой и артикулом. Оружие делится на огнестрельное и холодное. Холодное оружие характеризуется…

3.3 Проектирование программного модуля

3.3.1 Проектирование структур данных. Приводится описание используемых структур данных. К рассматриваемым структурам относятся классы, массивы объектов классов, массивы с элементами базовых типов, списки и т. д. Для приведенных структур указывается назначение в программе.

В этом же пункте приводится диаграмма классов. Диаграмма классов должна отражать классы, свойства, методы, связи между классами. Для построения можно использовать возможности MS Visual Studio, пакеты моделирования или графические приложения.

Для каждого класса строится таблица с указанием полей, их типов, ограничений на значения.

Пример.

Таблица 2.1 – Описание полей класса Товар

Имя поля	*Тип данных*	*Ограничение*
Наименование	*string*	*Длина от 10 до 255 символов*
Артикул	*string*	*Длина 12 символов, может включать латинский алфавит и цифры*
Цена	*float*	*Положительное*

При проектировании классов разработанные однотипные функции, например функции поиска, следует включить в отдельные специальные классы, а обрабатываемые структуры, например массивы объектов, передавать в качестве параметров. Такие классы также описываются в данном пункте.

3.3.2 Проектирование взаимодействия с пользователем. Строится диаграмма взаимодействия, отражающая обращения пользователя, вводимые им данные и передачу этих данных классам для обработки. Для построения используется средство, выбранное в предыдущем пункте для диаграммы классов.

3.3.3 Проектирование интерфейса. Приводится описание форм и используемых элементов, изображение форм в режиме проектирования. Для консольного

интерфейса приводится структура меню в виде иерархии, указываются вызываемые в пунктах функции. Пример фрагмента структуры меню представлен на рисунке 3.1.

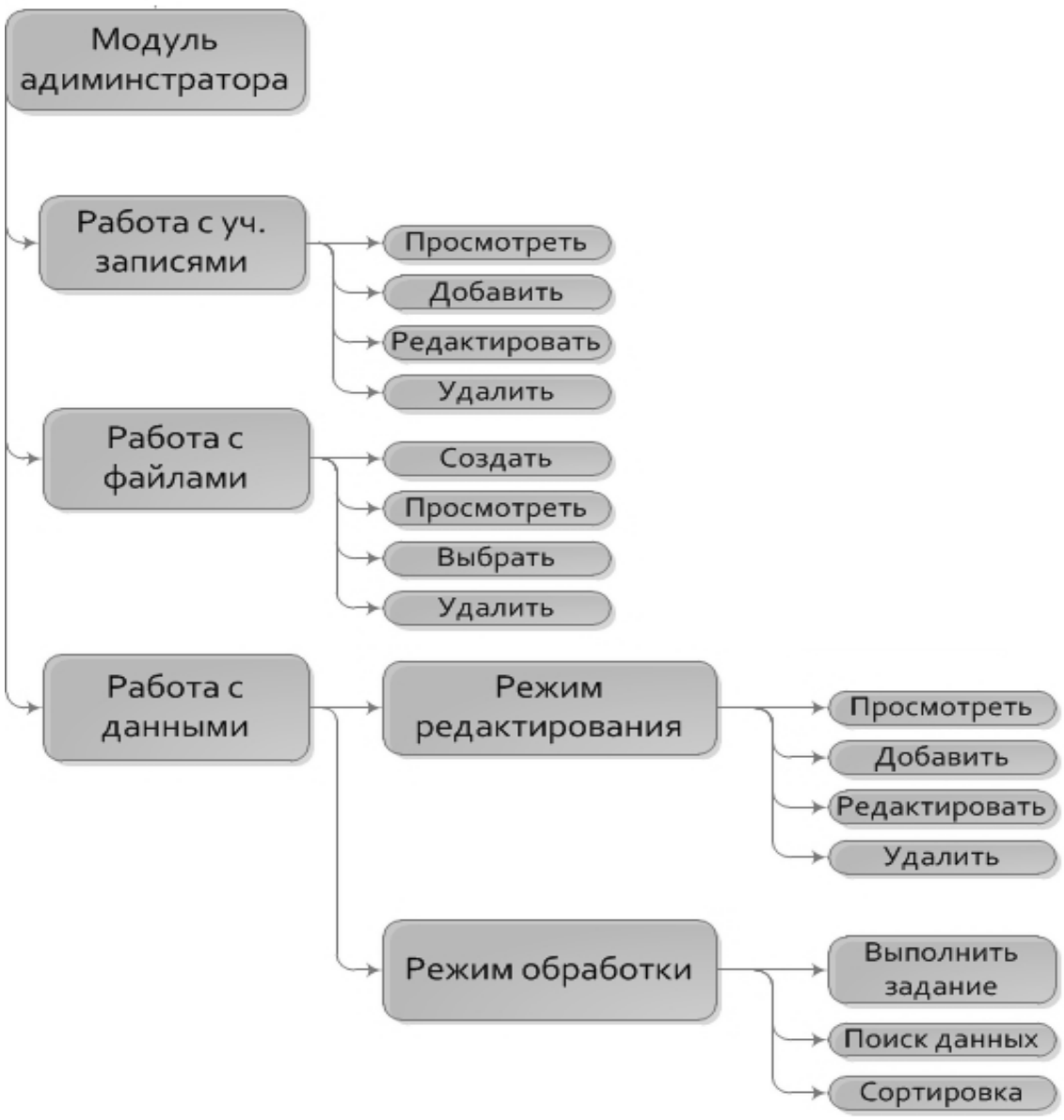


Рисунок 3.1 – Пример структуры меню

3.4 Реализация программного модуля

3.4.1 Реализация классов и структур данных. Приводится программный код и словесные пояснения к рассмотренным в подразд. 2.2 классам и структурам данных.

3.4.2 Реализация требований. Для всех требований к функциям программного средства, описанных в подразд. 1.1 строится таблица, в которой для каждого требования указывается его реализация (имя реализующей функции или класс и реализующий метод).

Пример.

Таблица 2.2 – Соответствие требований реализации

Функциональное требование	*Реализация*	*Примечание*
Должна производиться сортировка товаров по артикулу в порядке возрастания	*Метод SortByArt класса Sort*	*Параметр – массив объектов класса product*
Должна производиться сортировка товаров по наименованию в порядке возрастания	*Метод SortByName класса Sort*	*Параметр – массив объектов класса product*

3.4.2 Реализация концепций объектно-ориентированного программирования. Приводится определение концепций объектно-ориентированного программирования, таких как наследование, полиморфизм, абстракция и инкапсуляция. Для каждой из них приводятся краткое словесное описание реализации и фрагменты программного кода.

Пример.

Виртуальные функции поддерживают концепцию полиморфизма. В данном проекте разработаны следующие виртуальные функции…

Пример.

Для реализации инкапсуляции все поля классов имеют спецификатор доступа private.

3.4.3 Описание структуры проекта. Перечисляются созданные .cpp- и .h-файлы. Указывается, какие структурные единицы проекта в каких файлах реализованы. Указываются связи между файлами.

3.4.4 Описание интерфейса взаимодействия с пользователем. Приводятся снимки экрана с основными элементами интерфейса (формы в режиме работы или экран с различными пунктами меню), дается кратное словесное пояснение.

3.5 Тестирование программного модуля

В разделе «Выполнение программы» должна быть указана последовательность действий пользователя, обеспечивающих загрузку, запуск, выполнение всех функций программы. Выполнение каждой функции сопровождается снимком экрана с соответствующим результатом работы программы.

В документе исключено прямое обращение к пользователю. Должны отсутствовать слова «откройте», «нажмите», «укажите» и пр. Следует применять штампы «следует открыть», «следует нажать» и им подобные.

3.6 Заключение

В разделе приводятся результаты проделанной работы и делаются выводы о том, достигнута ли цель проектирования и решены ли поставленные задачи.

3.7 Список использованных источников

В списке перечисляются книги, статьи, источники из интернета, которые были использованы при выполнении работы. Информация о правилах оформления этого списка представлена в подразд. 4.8, а также в ГОСТ 7.1–2003 Библиографическая запись. Библиографическое описание. Общие требования и правила.

Список использованных источников должен содержать не менее пяти наименований не старше 10 лет.

3.8 Приложения

Приложения содержат материалы вспомогательного характера: алгоритмы, тексты программ, результаты тестирования, большие таблицы и т. д. В записке по данному курсовому проекту обязательным является наличие приложения с текстом программы. Наличие комментариев к программе обязательно. Правила оформления представлены в подразд. 4.7.

4 Оформление пояснительной записки

4.1 Общие требования

Пояснительная записка к курсовой работе оформляется в соответствии с правилами оформления текстовых документов, изложенными в ГОСТ 2.105–95 Общие требования к текстовым документам, и с правилами оформления курсовых работ, изложенными в данных методических рекомендациях. Список использованных источников оформляется в соответствии с правилами, изложенными в ГОСТ 7.1–2003.

Записка должна быть напечатана на одной стороне листа белой бумаги формата А4. Рекомендуется: шрифт – Times New Roman, размер шрифта – 13, межстрочный интервал – 1,15.

Отступ от рамки – не менее 5 мм.

Цвет шрифта – черный, при необходимости что-либо вписать следует использовать чернила, пасту или тушь черного цвета. Опечатки не зачеркивать, а заклеивать, подчищать или закрашивать белой краской.

В тексте после знаков препинания обязательно ставится пробел. Нельзя сокращать слова (кроме сокращений, установленных правилами орфографии). Например, пишется целиком «то есть», «так как». Нельзя употреблять специальные знаки типа «=», «%», «+» и другие без цифр.

Записка оформляется на листах с рамкой.

Раздел «Содержание» печатается на листе с основной подписью для заглавного листа (пример заполнения представлен на рисунке 4.1), остальные листы курсовой работы выполняются на листах с основной подписью для последующих листов (пример заполнения представлен на рисунке 4.2).

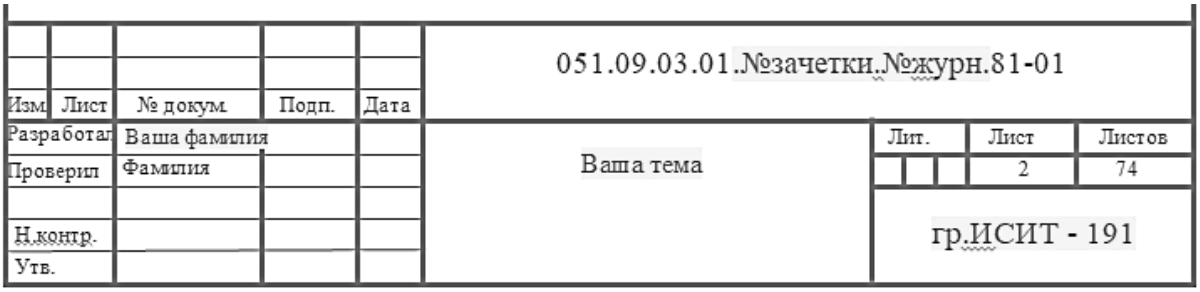


Рисунок 4.1 – Основная надпись для заглавного листа

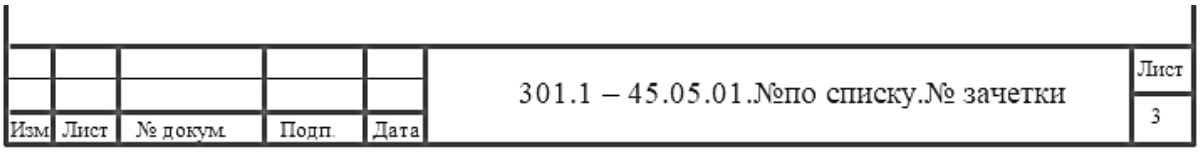


Рисунок 4.2 – Основная надпись для последующих листов

4.2 Нумерация страниц

Нумерация страниц сквозная, по всем страницам записки, включая приложения. Первая страница – это титульный лист, на нем номер не ставится. Номер проставляется арабской цифрой без точки в нижней части листа справа.

4.3 Построение текста

Рассмотрим разбивку и нумерацию структурных единиц записки. Структурные единицы пояснительной записки приведены в подразд. 2.1. Каждая структурная единица записки начинается с нового листа.

Структурные единицы записки «Содержание», «Введение», «Заключение», «Список использованных источников» не нумеруются. Соответствующие заголовки записываются обычным шрифтом с прописной буквы с выравниванием по центру. Заголовки не подчеркиваются и точка в конце них не ставится.

Текст основной части делят на разделы (номер раздела состоит из одной цифры), разделы – на подразделы (номер подраздела состоит из двух цифр), подразделы – на пункты и т. д.

Разделы нумеруются арабскими цифрами без точки; подразделы – в пределах раздела. Номер состоит из номера раздела и подраздела, разделенных точкой. В конце номера подраздела точка не ставится. За номером раздела или подраздела следует его название, оно записывается обычным шрифтом с прописной буквы без точки в конце. Переносы слов в заголовках не допускаются. Если заголовок состоит из двух предложений, их разделяют точкой.

Каждый раздел начинается с нового листа, подразделы с нового листа не начинаются. Номер и название раздела или подраздела записываются с абзацного отступа. Расстояние между заголовком и текстом, заголовками раздела и подраздела – одна пустая строка.

4.4 Рисунки

Все иллюстрации (чертежи, схемы, графики, структурные схемы и др. (кроме таблиц)) называются рисунками. Они должны располагаться непосредственно после ссылки на них в тексте или на следующей странице. Рисунок располагается так, чтобы его удобно было смотреть без поворота листа или с поворотом по часовой стрелке.

Рисунок центрируется, подпись располагается с абзацного отступа.

Рисунки, за исключением рисунков приложений, следует нумеровать арабскими цифрами сквозной нумерацией. Если рисунок один, то он обозначается, например, «Рисунок 1». Слово «Рисунок» и его номер располагаются с абзацного отступа.

Рекомендуется нумеровать иллюстрации в пределах разделов. В этом случае номер иллюстрации состоит из номера раздела и порядкового номера иллюстрации, разделенных точкой. Например, надпись «Рисунок 1.2» означает второй рисунок первого раздела.

Иллюстрации каждого приложения обозначают отдельной нумерацией арабскими цифрами, добавляя перед цифрой обозначение приложения. Например, «Рисунок А.3», что обозначает третий рисунок в приложении А.

Иллюстрации должны иметь наименование и пояснительные данные (подрисуночный текст). Порядок расположения следующий: вначале – сам рисунок, затем – подрисуночный текст, далее – слово «Рисунок», номер рисунка и наименование в виде, например, «Рисунок 1 – Структурная схема».

4.5 Таблицы

В тексте пояснительной записки следует помещать итоговые и наиболее важные таблицы, таблицы вспомогательного и справочного характера – в приложениях.

Таблицы нумеруются и обозначаются по тем же правилам, что и иллюстрации (см. подразд. 3.4), только вместо слова «Рисунок» пишется слово «Таблица», это слово и название таблицы располагаются без абзацного отступа над таблицей. Примеры обозначений: «Таблица 1», «Таблица В.1», «Таблица 3.1 – Список модулей».

Таблицы ограничиваются одинарными горизонтальными и вертикальными линиями. Использование диагональных линий не допускается.

Если таблица выходит за формат листа, то ее делят на части. Части таблицы можно помещать на одном листе одна над другой или рядом либо переносить на другие листы. При переносе части таблицы на другой лист заголовок помещают только над первой частью, то есть слово «Таблица» с названием указывают только над первой частью, а над другими частями пишут слова «Продолжение

таблицы» с указанием номера таблицы. Шапку таблицы при переносе части таблицы не повторяют, переносится строка с номерами столбцов.

4.6 Ссылки

Примеры ссылок в тексте пояснительной записки:
1) на иллюстрацию – «… в соответствии с рисунком 1.2 …»;
2) на формулу - «… в формуле (2.1) …»;
3) на приложение - «… (приложение Б) …»;
4) на таблицу - «…в таблице 1.2 …».

Повторные ссылки следует давать с сокращенным словом «смотри», например: (см. рисунок 1.2), (см. приложение Б).

На материалы, взятые из литературы или других источников, должны быть даны ссылки с указанием номера источника по списку использованных источников (см. подразд. 3.9). Номер ссылки проставляется арабскими цифрами в квадратных скобках, например: [1], [1, 2, 5].

4.7 Приложения

Каждое приложение начинается с нового листа с указанием наверху посередине слова «Приложение» с первой прописной буквы и его обозначения. Приложения обозначаются заглавными буквами русского алфавита, начиная с А, за исключением букв Ё, З, Й, О, Ч, Ь , Ы, Ъ. Например, «приложение А», «приложение Б» и т. д.

После строки, содержащей слово «Приложение», то есть на второй строке, записывается симметрично по тексту с первой прописной буквы заголовок приложения. Например:

Приложение А
Текст программы

Затем через одну пустую строчку следует текст приложения. Текст каждого приложения может быть разделен на разделы, подразделы и т. д., которые нумеруются арабскими цифрами в пределах приложения по аналогии с разделами и подразделами основной части пояснительной записки. Например, «П. А. 2» – это обозначение второго раздела приложения А.

Код программы допускается оформлять шрифтом меньшего размера до 10 пт.

4.8 Список использованных источников

Источники – это книги, учебники, диссертации, статьи из журналов, статьи из интернета и т. д., использованные при выполнении курсовой работы. Источники в списке располагаются в порядке ссылок в тексте записки или по алфавиту, нумеруются арабскими цифрами без точки и печатаются с абзацного отступа,

при этом дается библиографическое описание каждого источника в соответствии с ГОСТ 7.1, ГОСТ 7.12.

Общий шаблон описания книги, у которой не более трех авторов: ФИО автора, название книги, точка, тире, город, двоеточие, издательство, запятая, год издания, точка, тире, количество страниц, буква «с», точка.

Название города дается целиком, допустимы только сокращения «М.» (Москва) и «СПб.» (Санкт-Петербург); название издательства – без кавычек. Если у книги один, два или три автора, то вначале указывается фамилия, потом – инициалы.

Пример.

Один, *два или три автора*:

Шуп, Т. Решение инженерных задач на ЭВМ: практическое руководство: пер. с англ. / Т. Шуп. – Москва: Мир, 1982. – 238 с.

Четыре и более автора:

Зубчатые передачи: справочник / Е. Г. Гинзбург [и др.]; под ред. Н. Т. Халебского. – Минск: Машиностроение, 1980. – 416 с.

Книги под общей редакцией:

Шейнин, А. М. Эксплуатация дорожных машин / А. М. Шейнин; под общ. ред. А. М. Шейнина. – Москва: Транспорт, 1992. – 328 с.

Стандарты:

ГОСТ 19.701–90. Схемы алгоритмов, программ данных и систем. – Москва: Изд-во стандартов, 2004. – 26 с.

Список литературы

1 **Страуструп, Б.** Программирование: принципы и практика с использованием C++ / Б. Страуструп. – Москва: Вильямс, 2016. – 1328 с.

2 **Боровский, А. Н.** Qt4.7+. Практическое программирование на C++. / А. Н. Боровский. – Санкт-Петербург: BHV, 2012. – 496 с.

3 **Васильев, А. Н.** Объектно-ориентированное программирование на C++ / А. Н. Васильев. – Санкт-Петербург: Наука и техника, 2016. – 544 с.

4 **Васильев, А. Н.** Программирование на C++ в примерах и задачах / А. Н. Васильев. – Москва: ЭКСМО, 2017. – 416 с.

5 **Кёниг, Э.** Эффективное программирование на C++. Практическое программирование на примерах. Серия «C++ In-Depth» / Э. Кёниг, Б. Му – Москва: Диалектика, 2019. – 368 с.

6 **Кениг, Э.** Эффективное программирование на C++. Практическое программирование на примерах / Э. Кёниг, Б. Му – Москва: Вильямс, 2016. – 368 с.

7 **Лафоре, Р.** Объектно-ориентированное программирование в C++. Классика Computer Science / Р. Лафоре. – Санкт-Петербург: Питер, 2013. – 928 с.

8 **МакГрат, М.** Программирование на C для начинающих / М. МакГрат. – Москва: Эксмо, 2015. – 192 с.

9 **Мартин Р.** Чистый код. Создание, анализ и рефакторинг. Библиотека программиста / Р. Мартин. – Санкт-Петербург: Питер, 2018. – 464 с.

10 **Мейерс, С.** Эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов / С. Мейерс. – Москва: ДМК Пресс, 2014. – 294с.

11 **Павловская, Т. А.** C/C++. Программирование на языке высокого уровня: учебник для вузов / Т. А. Павловская. – Санкт-Петербург: Питер, 2016. – 461 с.

12 **Полубенцева, М.** C/C++. Процедурное программирование / М. С. Полубенцева. – Санкт-Петербург: BHV, 2017. – 432 с.

13 **Перри, Г.** Программирование на C для начинающих / Г. Перри, Д. Миллер. – Москва: ЭКСМО, 2016. – 192 с.

14 **Хенкеманс, Д.** Программирование на C++ / Д. Хенкеманс, М. Ли. – Санкт-Петербург: Символ-плюс, 2015. – 416 с.

15 **Шилдт, Г. С.**++. Базовый курс / Г. Шилдт. – Москва: Вильямс, 2015. – 624 с.

Приложение А
(обязательное)

Примерный перечень заданий на курсовое проектирование

1 Создать шаблон класса «стек». Написать программу, использующую этот шаблон класса для моделирования Т-образного сортировочного узла на железной дороге. Программа должна разделять на два направления состав, состоящий из вагонов двух типов (на каждое направление формируется состав из вагонов одного типа).

2 Создать шаблон класса «стек». Написать программу, использующую этот шаблон для отыскания прохода по лабиринту с применением данного шаблона класса.

3 Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице. После отыскания прохода программа печатает найденный путь в виде координат квадратов.

4 Создать шаблон класса «бинарное дерево». Использовать его для сортировки целых чисел и строк, задаваемых с клавиатуры или из файла.

5 Создать шаблон класса «очередь». Написать программу, демонстрирующую работу с этим шаблоном для различных типов параметров шаблона. Программа должна содержать меню, позволяющее осуществить проверку всех методов шаблона.

6 Создать шаблон класса «очередь с приоритетами». При добавлении элемента в такую очередь его номер определяется его приоритетом. Написать программу, демонстрирующую работу с этим шаблоном для различных типов параметров шаблона. Программа должна содержать меню, позволяющее осуществить проверку всех методов шаблона.

7 Создать шаблон класса «бинарное дерево», обладающее возможностью добавления новых элементов, удаления существующих, поиска элемента по ключу, а также последовательного доступа ко всем элементам.

8 Написать программу, использующую этот шаблон для представления англо-русского словаря. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса. Предусмотреть возможность формирования словаря из файла и с клавиатуры.

9 Разработать программное средство для учета данных о товарах в аптеке. Программа должна содержать меню, позволяющее осуществить редактирование ассортимента, поиск и сортировку по различным критериям.

10 Разработать программное средство для учета данных о товарах в аптеке. Программа должна содержать меню, позволяющее осуществить редактирование ассортимента, поиск и сортировку по различным критериям.

Приложение Б
(обязательное)

Образец оформления титульного листа

МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ТЕМА КУРСОВОГО ПРОЕКТА

Курсовой проект
по дисциплине «Объектно-ориентированное программирование»

301.1 – 45.05.01. № по списку. № зачетки

Исполнитель _____ ФИО студента, группа
(подпись)

Руководитель _____ ФИО руководителя
(подпись)

Дата допуска к защите _____

Дата защиты _____

Оценка _____

Могилев 2021

Приложение В
(справочное)

Правила оформления программного кода на языке C++

1 Соглашения об именовании

1.1 Общие соглашения об именовании

1.1.1 Имена, представляющие типы, должны быть обязательно написаны в смешанном регистре, начиная с верхнего.

Line, SavingsAccount

Общая практика в сообществе разработчиков C++.

1.1.2 Имена переменных должны быть записаны в смешанном регистре, начиная с нижнего.

line, savingsAccount

Общая практика в сообществе разработчиков C++. Позволяет легко отличать переменные от типов, предотвращает потенциальные коллизии имён, например:

Line line;

1.1.3 Именованные константы (включая значения перечислений) должны быть записаны в верхнем регистре с нижним подчёркиванием в качестве разделителя.

MAX_ITERATIONS, COLOR_RED, PI

Общая практика в сообществе разработчиков C++. Использование таких констант должно быть сведено к минимуму. В большинстве случаев реализация значения в виде метода – лучшее решение:

```
int getMaxIterations() // НЕЛЬЗЯ: MAX_ITERATIONS = 25
{
  return 25;
}
```

Эта форма более читаемая и гарантирует единый интерфейс к значениям, хранящимся в классе.

1.1.4 Названия методов и функций должны быть глаголами, быть записанными в смешанном регистре и начинаться с нижнего.

getName(), computeTotalWidth()

Совпадает с правилом для переменных, но отличие между ними состоит в их специфических формах.

1.1.5 Названия пространств имён следует записывать в нижнем регистре.

model::analyzer, io::iomanager, common::math::geometry

Общая практика в сообществе разработчиков C++.

1.1.6 Следует называть имена типов в шаблонах одной заглавной буквой.

```
template<class T> ...
template<class C, class D> ...
```

Общая практика в сообществе разработчиков C++. Позволяет выделить имена шаблонов среди других используемых имён.

1.1.7 Аббревиатуры и сокращения в именах должны записываться в нижнем регистре.

```
exportHtmlSource(); // НЕЛЬЗЯ: exportHTMLSource();
openDvdPlayer();    // НЕЛЬЗЯ: openDVDPlayer();
```

Использование верхнего регистра может привести к конфликту имён, описанному выше. Иначе переменные имели бы имена dVD, hTML и т. д., что не является удобочитаемым. Другая проблема уже описана выше; когда имя связано с другим, читаемость снижается; слово, следующее за аббревиатурой, не выделяется так, как следовало бы.

1.1.8 Глобальные переменные всегда следует использовать с оператором разрешения области видимости (::).

::mainWindow.open(), ::applicationContext.getName()

Следует избегать использования глобальных переменных. Предпочтительнее использование синглтонов.

1.1.9 Членам класса с модификатором private следует присваивать суффикс-подчёркивание.

```
class SomeClass {
  private:
    int length_;
}
```

Кроме имени и типа, область видимости – наиболее важное свойство переменной. Явное указание модификатора доступа в виде подчёркивания избавляет от путаницы между членами класса и локальными переменными. Это важно, поскольку переменные класса имеют большее значение, нежели переменные в методах, и к ним следует относиться более осторожно.

Дополнительным эффектом от суффикса-подчёркивания является разрешение проблемы именования в методах, устанавливающих значения, а также в конструкторах:

```
  void setDepth (int depth)
  {
    depth_ = depth;
  }
```

Проблема заключается в том, что существует два варианта подчёркивания – в виде суффикса и в виде префикса. Оба варианта широко используются, но рекомендуется именно первый вариант, потому что он обеспечивает лучшую читаемость.

1.1.10 Настраиваемым переменным следует давать то же имя, что и у их типа.

```
void setTopic(Topic* topic)      // НЕЛЬЗЯ: void setTopic(Topic* value)
                                 // НЕЛЬЗЯ: void setTopic(Topic* aTopic)
                                 // НЕЛЬЗЯ: void setTopic(Topic* t)

void connect(Database* database) // НЕЛЬЗЯ: void connect(Database* db)
                                 // НЕЛЬЗЯ: void connect (Database* oracleDB)
```

Сокращайте сложность путём уменьшения числа используемых терминов и имён. Также упрощает распознавание типа просто по имени переменной.

Если по какой-то причине эта рекомендация кажется неподходящей, это означает, что имя типа выбрано неверно.

Не являющиеся настраиваемыми переменные могут быть названы по их назначению и типу:

```
  Point  startingPoint, centerPoint;
  Name   loginName;
```

1.1.11 Все имена следует записывать по-английски.

```
fileName;   // НЕ РЕКОМЕНДУЕТСЯ: imyaFayla
```

Английский наиболее предпочитетелен для интернациональной разработки.

1.1.12 Переменные, имеющие большую область видимости, следует называть длинными именами, имеющие небольшую область видимости – короткими.

Имена временных переменных, использующихся для хранения временных значений или индексов, лучше всего делать короткими. Программист, читающий такие переменные, должен иметь возможность предположить, что их значения не используются за пределами нескольких строк кода. Обычно это переменные *i*, *j*, *k*, *l*, *m*, *n* (для целых), а также *c* и *d* (для символов).

1.1.13 Имена объектов не указываются явно, следует избегать указания названий объектов в именах методов.

```
line.getLength();   // НЕ РЕКОМЕНДУЕТСЯ: line.getLineLength();
```

Второй вариант смотрится вполне естественно в объявлении класса, но совершенно избыточен при использовании, как это и показано в примере.

1.2 Особые правила именования

1.2.1 Слова **get/set** должны быть использованы везде, где осуществляется прямой доступ к атрибуту.

```
employee.getName();
employee.setName(name);

matrix.getElement(2, 4);
matrix.setElement(2, 4, value);
```

Общая практика в сообществе разработчиков C++.

1.2.2 Слово **compute** может быть использовано в методах, вычисляющих что-либо.

```
valueSet->computeAverage();
matrix->computeInverse()
```

1.2.3 Слово find может быть использовано в методах, осуществляющих какой-либо поиск.

```
vertex.findNearestVertex();
matrix.findMinElement();
```

1.2.4 Слово initialize может быть использовано там, где объект или сущность инициализируется.

```
printer.initializeFontSet();
```

1.2.5 Множественное число следует использовать для представления наборов (коллекций) объектов.

```
vector<Point>  points;
int            values[];
```

Улучшает читаемость, поскольку имя даёт пользователю прямую подсказку о типе переменной и операциях, которые могут быть применены к этим элементам.

1.2.6 Префикс **n** следует использовать для представления числа объектов.

nPoints, nLines

1.2.7 Суффикс **No** следует использовать для обозначения номера сущности.

tableNo, employeeNo

Другой неплохой альтернативой является префикс *i*: iTable, iEmployee. Он ясно даёт понять, что перед нами именованный итератор.

1.2.8 Префикс **is** следует использовать только для булевых (логических) переменных и методов.

isSet, isVisible, isFinished, isFound, isOpen

Общая практика в сообществе разработчиков C.

Использование этого префикса избавляет от таких имён, как *status* или *flag*. *isStatus* или *isFlag* просто не подходят, и программист вынужден выбирать более осмысленные имена.

1.2.9 Симметричные имена должны использоваться для соответствующих операций.

get/set, add/remove, create/destroy, start/stop, insert/delete,

increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, и т. д.

1.2.10 Следует избегать сокращений в именах.

```
computeAverage();   // НЕЛЬЗЯ: compAvg();
```

1.2.11 Классам исключений следует присваивать суффикс **Exception.**

```
class AccessException
{
  :
}
```

Классы исключений в действительности не являются частью архитектуры программ, и такое именование отделяет их от других классов.

1.2.12 Функциям (методам, возвращающим какие-либо значения) следует давать имена в зависимости от того, что они возвращают, а процедурам – в зависимости от того, что они выполняют (методы void).

2 Файлы

2.1 Файлы исходных кодов

2.1.1 Заголовочным файлам C++ следует давать расширение **.h** (предпочтительно) либо **.hpp**. Файлы исходных кодов могут иметь расширения **.c++** (рекомендуется), **.C, .cc** либо **.cpp**.

MyClass.c++, MyClass.h

Это расширения, одобряемые стандартом C++.

2.1.2 Класс следует объявлять в заголовочном файле и определять (реализовывать) в файле исходного кода, имена файлов совпадают с именем класса.

MyClass.h, MyClass.c++

Облегчает поиск связанных с классом файлов. Очевидное исключение – шаблонные классы, которые должны быть объявлены и определены в заголовочном файле.

2.1.3 Все определения должны находиться в файлах исходного кода.

```
class MyClass
{
public:
  int getValue () {return value_;}  // НЕЛЬЗЯ!
  ...

private:
  int value_;
}
```

Заголовочные файлы объявляют интерфейс, файлы исходного кода его реализовывают. Если программисту необходимо найти реализацию, он должен быть уверен, что найдёт её именно в файле исходного кода.

2.1.4 Незавершённость разбитых строк должна быть очевидна.

```
totalSum = a + b + c +
           d + e;

function (param1, param2,
          param3);

setText ("Long line split"
         "into two parts.");

for (int tableNo = 0; tableNo < nTables;
     tableNo += tableStep) {
  ...
}
```

2.2 Включения файлов

2.2.1 Заголовочные файлы должны содержать защиту от вложенного включения.

```
#ifndef COM_COMPANY_MODULE_CLASSNAME_H
#define COM_COMPANY_MODULE_CLASSNAME_H
  :
#endif // COM_COMPANY_MODULE_CLASSNAME_H
```

Конструкция позволяет избегать ошибок компиляции. Это соглашение позволяет увидеть положение файла в структуре проекта и предотвращает конфликты имён.

2.2.2 Директивы включения следует сортировать (по месту в иерархии системы, ниже уровень – выше позиция) и группировать. Оставляйте пустую строку между группами.

```
#include <fstream>
#include <iomanip>

#include <qt/qbutton.h>
#include <qt/qtextfield.h>

#include "com/company/ui/PropertiesDialog.h"
#include "com/company/ui/MainWindow.h"
```

Пути включения не должны быть абсолютными. Вместо этого следует использовать директивы компилятора.

2.2.3 Директивы включения должны располагаться только в начале файла.

Общая практика. Избегайте нежелательных побочных эффектов, которые может вызвать «скрытое» включение где-то в середине файла исходного кода.

3 Выражения

3.1 Типы

3.1.1 Локальные типы, используемые в одном файле, должны быть объявлены только в нём.

Улучшает сокрытие информации.

3.1.2 Разделы класса public, protected и private должны быть отсортированы. Все разделы должны быть явно указаны.

Сначала должен идти раздел *public*, что избавит желающих ознакомиться с классом от чтения разделов *protected/private*.

3.1.3 Приведение типов должно быть явным. Никогда не полагайтесь на неявное приведение типов.

```
floatValue = static_cast<float>(intValue); // НЕЛЬЗЯ: floatValue = intValue;
```

Этим программист показывает, что ему известно о различии типов, что смешение сделано намеренно.

3.2 Переменные

3.2.1 Следует инициализировать переменные в месте их объявления.

Это даёт гарантию, что переменные пригодны для использования в любой момент времени. Но иногда нет возможности осуществить это:

```
int x, y, z;
getCenter(&x, &y, &z);
```

В этих случаях лучше оставить переменные неинициализированными, чем присваивать им какие-либо значения.

3.2.2 Следует избегать использования глобальных переменных.

Не существует причины использовать глобальные переменные в C++. То же касается глобальных функций и (статических) переменных, область видимости которых – весь файл.

3.2.3 Не следует объявлять переменные класса как **public**.

Эти переменные нарушают принципы сокрытия информации и инкапсуляции. Вместо этого используйте переменные с модификатором *private* и соответствующие функции доступа. Исключение – класс без поведения, практически структура данных (эквивалент структур языка C). В этом случае нет смысла скрывать эти переменные.

3.2.4 Символ указателя или ссылки в языке C++ следует ставить сразу после имени типа, а не с именем переменной.

```
float* x; // НЕ РЕКОМЕНДУЕТСЯ: float *x;
int& y;   // НЕ РЕКОМЕНДУЕТСЯ: int &y;
```

То, что переменная – указатель или ссылка, относится скорее к её типу, а не к имени. Программисты на C часто используют другой подход, но в C++ лучше придерживаться этой рекомендации.

3.3 Разное

3.3.1 Следует избегать «магических» чисел в коде. Числа, отличные от 0 или 1, следует объявлять как именованные константы.

Если число само по себе не имеет очевидного значения, читаемость улучшается путём введения именованной константы. Другой подход – создание метода, с помощью которого можно было бы осуществлять доступ к константе.

3.3.2 Константы с плавающей точкой следует записывать с десятичной точкой и с указанием по крайней мере одной цифры после запятой.

```
double total = 0.0;    // НЕ РЕКОМЕНДУЕТСЯ:  double total = 0;
double speed = 3.0e8;  // НЕ РЕКОМЕНДУЕТСЯ:  double speed = 3e8;

double sum;
:
sum = (a + b) * 10.0;
```

Это подчёркивает различные подходы при работе с целыми числами и числами с плавающей точкой. С точки зрения математики эти две модели совершенно различны и не совместимы.

3.3.3 Константы с плавающей точкой следует всегда записывать, по крайней мере, с одной цифрой до десятичной точки.

```
double total = 0.5;  // НЕ РЕКОМЕНДУЕТСЯ:  double total = .5;
```

Система чисел и выражений в C++ заимствована из математики, и следует придерживаться традиционных форм записи, где это возможно. Помимо прочего, 0.5 — более читаемо, чем .5 (первый вариант никак не спутать с числом 5).

3.3.4 У функций нужно обязательно указывать тип возвращаемого значения.

```
int getValue()   // НЕЛЬЗЯ: getValue()
{
  :
}
```

Если это не указано явно, C++ считает, что возвращаемое значение имеет тип int. Никогда нельзя полагаться на это, поскольку такой способ может смутить программистов, не знакомых с ним.

3.3.5 Не следует использовать **goto.**

Этот оператор нарушает принципы структурного программирования. Следует использовать только в очень редких случаях (например, для выхода из глубоко вложенного цикла), когда иные варианты однозначно ухудшат читаемость.

4 Оформление и комментарии

4.1 Оформление

4.1.1 Основной отступ следует делать в два пробела.

```
for (i = 0; i < nElements; i++)
  a[i] = 0;
```

Отступ в один пробел достаточно мал, чтобы отражать логическую структуру кода. Отступ более четырех пробелов делает глубоко вложенный код нечитаемым и увеличивает вероятность того, что строки придётся разбивать. Широко распространены варианты в два, три или четыре пробела; причём два и четыре – более широко.

4.1.2 Блоки кода следует оформлять так, как показано в примере 1 (рекомендуется) или в примере 2, но ни в коем случае не так, как показано в примере 3. Оформление функций и классов должно следовать примеру 2.

Пример 1

```
while (!done) {
  doSomething();
  done = moreToDo();
}
```

Пример 2

```
while (!done)
{
  doSomething();
  done = moreToDo();
}
```

Пример 3

```
while (!done)
  {
    doSomething();
    done = moreToDo();
  }
```

Пример 3 использует лишние отступы, что мешает ясному отображению логической структуры кода.

4.1.3 Объявления классов следует оформлять следующим образом:

```
class SomeClass : public BaseClass
{
  public:
    ...

  protected:
    ...

  private:
    ...
}
```

Частное следствие из правила, указанного выше.

4.1.4 Определения методов следует оформлять следующим образом:

```
void someMethod()
{
  ...
}
```

Следствие из правила, указанного выше.

4.1.5 Конструкцию if-else следует оформлять следующим образом:

```
if (condition) {
  statements;
}

if (condition) {
  statements;
}
else {
   statements;
}

if (condition) {
  statements;
}
else if (condition) {
  statements;
}
else {
  statements;
}
```

Следствие из правила, указанного выше. Причём написание *else* на той же строке, где стоит закрывающая фигурная скобка первого блока, не является запрещённым:

```
if (condition) {
  statements;
} else {
  statements;
}
```

Лучше каждую часть if-else помещать на отдельной строке. Это упрощает действия с кодом, например, перемещение блока *else*.

4.1.6 Цикл for следует оформлять следующим образом:

```
for (initialization; condition; update) {
  statements;
}
```

Следствие из правила, указанного выше.

4.1.7 Цикл for с пустым телом следует оформлять следующим образом:

```
for (initialization; condition; update)
  ;
```

Делает акцент для читающего на том, что тело пусто. Однако циклов, не имеющих тела, следует избегать.

4.1.7 Цикл while следует оформлять следующим образом:

```
while (condition) {
  statements;
}
```

Следствие из правила, указанного выше.

4.1.8 Цикл do-while следует оформлять следующим образом:

```
do {
  statements;
} while (condition);
```

Следствие из правила, указанного выше.

4.1.9 Конструкцию switch следует оформлять следующим образом:

```
switch (condition) {
  case ABC :
    statements;
    // Отсутствует "break"

  case DEF :
    statements;
    break;

  case XYZ :
    statements;
    break;

  default :
    statements;
    break;
}
```

Обратите внимание, что каждое слово *case* имеет отступ относительно всей конструкции, что помогает её выделить. Также обратите внимание на пробелы перед двоеточиями. Если где-то отсутствует ключевое слово *break*, то предупреждением об этом должен служить комментарий. Программисты часто забывают ставить это слово, поэтому случай нарочного его пропуска должен описываться специально.

4.1.10 Конструкцию try-catch следует оформлять следующим образом:

```
try {
  statements;
}
catch (Exception& exception) {
  statements;
}
```

Следствие из правила, указанного выше. Вопросы, касающиеся закрывающих фигурных скобок у конструкции *if-else*, применимы и здесь.

4.1.11 Если конструкция if-else содержит только одно выражение в теле, фигурные скобки можно опускать.

```
if (condition)
  statement;
```

```
while (condition)
  statement;

for (initialization; condition; update)
  statement;
```

Рекомендуется всё же не опускать фигурные скобки.

4.1.12 Возвращаемый функцией тип может располагаться над именем самой функции.

```
void
MyClass::myMethod(void)
{
  :
}
```

Так функции выровнены в одну колонку.

4.2 Пробелы

4.2.1 Операторы следует отбивать пробелами.

После зарезервированных ключевых слов языка C++ следует ставить пробел.
После запятых следует ставить пробелы.
Двоеточия следует отбивать пробелами.
После точек с запятой в цикле *for* следует ставить пробелы.

4.2.2 Логические блоки в коде следует отделять пустой строкой.

Улучшает читаемость.

4.2.3 Методы рекомендуется отделять тремя пустыми строками.

Это позволяет лучше их выделять.

4.3 Комментарии

4.3.1 Сложный код, написанный с использованием хитрых ходов, следует не комментировать, а переписывать!

Следует делать как можно меньше комментариев, делая код самодокументируемым путём выбора правильных имён и создания ясной логической структуры.

4.3.2 Комментарии следует располагать так, чтобы они относились к тому, что они описывают.

```
while (true) {          // НЕ РЕКОМЕНДУЕТСЯ:  while (true) {
  // Do something                                    // Do something
  something();                                       something();
}                                                  }
```

Это делается с тем, чтобы избежать ситуации, когда комментарии нарушают логическую структуру программы.