МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
Казанский национальный исследовательский технический университет
им. А. Н. Туполева-КАИ
(КНИТУ-КАИ)
Альметьевский филиал

ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

Методические указания к выполнению курсовой работы

Альметьевск
2019

Методические указания к выполнению курсовой работы по дисциплине «Программирование на языке высокого уровня» содержат варианты заданий, краткие теоретические основы проектирования, разработки и тестирования прикладного программного обеспечения, правила оформления текстовой части работы и перечень необходимого и рекомендуемого учебно-методического обеспечения.

СОДЕРЖАНИЕ

1. Общие требования

Цель курсовой работы по дисциплине «Программирование на языке высокого уровня» состоит в закреплении и углублении знаний и навыков, полученных при изучении дисциплины. Курсовая работа предполагает выполнение задания повышенной сложности по проектированию, разработке и тестированию программного обеспечения, а также оформлению сопутствующей документации.

Отрасль современной разработки программного обеспечения (ПО) характеризуется тем, что это вид *коллективного* творчества, и создать что-либо серьезное в одиночку практически невозможно. При этом на коммуникации между участниками расходуется до половины общего времени работы над проектом, что позволяет сделать вывод о чрезвычайной значимости эффективной командной работы. Поэтому, для развития компетенций командного решения задач, выполнение задания по курсовой работе предлагается реализовать в группе из 2-3 исполнителей.

В качестве среды разработки следует использовать Borland C++ Builder, что позволит получить первичные навыки работы в современной и широко используемой на практике среде разработки приложений. Приложение следует создать консольного типа, что не предполагает использования средств быстрой разработки (*RAD*-средств) пользовательского интерфейса, обработчиков и т.п., что, в свою очередь, способствует более глубокому пониманию основ создания ПО.

1.1 Варианты заданий

1. **«Лабиринт».**

Программа автоматического и ручного прохождения роботом через лабиринт.

а) Реализовать модуль для создания лабиринта.

Реализовать проверку корректности созданного лабиринта (действительно ли в нем можно пройти от входа к выходу). Робот не может пройти сквозь стену. Сложность лабиринта (размер, количество стен и др.) задаются пользователем.

б) Реализовать модуль для управления мобильным роботом.

Робот может идти вперед, поворачиваться влево и вправо, разворачиваться. Робот обладает памятью на N последовательных ходов (N определяется пользователем).

в) Найти зависимость времени прохождения лабиринта от сложности лабиринта, глубины памяти робота.

2. **Игра «Судоку».**

Создать программу для генерации задач Судоку и их решения в автоматическом и ручном режимах.

а) Реализовать модуль для генерации задач. Размер поля и сложность определяются пользователем.

б) Реализовать модуль для решения задач.

в) Найти зависимость времени решения задачи от ее сложности для различных способов автоматического решения.

3. **Игра «Жизнь».**

Имеется поле, поделенное на клетки. Каждая клетка на этой поверхности может находиться в двух состояниях – «живая» или «мёртвая». Клетка имеет $N = ((2d+1)\times(2d+1) - 1)$ соседних клеток, где d – порядок соседства. Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение (шаг) рассчитывается на основе предыдущего по таким правилам:

- пустая («мёртвая») клетка с $N_ж$ живых клеток-соседей оживает;
- если у «живой» клетки есть $\geq(N_ж - 1)$ живых соседей, то эта клетка продолжает жить, если «живых» соседей меньше $(N_ж - 1)$ или больше $(N_ж + 1)$, то клетка «умирает» (от «одиночества» или от «перенаселённости»).

Игрок устанавливает правила взаимодействия клеток, расставляет «живые» клетки первого поколения, максимальное число итераций моделирования $Iter_{\max}$. Моделирование осуществляется автоматически до достижения $Iter_{\max}$ или если не осталось «живых» клеток.

а) Реализовать алгоритм игры, обрабатывающий взаимодействия клеток с возможностью применения уже существующей конфигурации (сгенерированной случайно, произвольно установленной пользователем или загруженной из файла).

б) Реализовать отображение состояния поля в текстовом режиме с возможностью ведения файла журнала, а также сохранения и загрузки:

- игрового поля на каждом шаге игры;
- конфигурации параметров моделирования.

в) Найти зависимость между правилами изменений популяции, размером окрестности d и количеством итераций моделирования.

4. **«Поиск оптимального маршрута».**

Реализовать программу поиска оптимального пути из одного пункта в другой с точки зрения выбранного критерия.

а) Реализовать модуль для генерации карты населенных пунктов и дорожной сети (граф). Автоматически и вручную задаются следующие параметры: количество вершин, плотность дорожной сети, направленность ребер

графа (однонаправленный, двунаправленный), вес ребер. Предусмотреть возможность сохранения и загрузки информации о графе.

б) Реализовать модуль для решения задачи поиска кратчайшего пути на графе между указанными пользователем вершинами. Кратчайший маршрут с перечислением пройденных вершин, длина найденного маршрута, время поиска решения показываются графически на экране (текстовый режим) и выводятся в файл отчета. Если решение не существует, то в файл отчета выводится соответствующее сообщение и время работы модуля.

в) Найти зависимость времени поиска оптимального маршрута от сложности графа.

5. **«Анализатор базы данных сотового оператора».**

Разработать программу анализа статистики телефонных звонков и сообщений в общем по базе данных и по каждому номеру в отдельности. В базе данных по каждому номеру содержится следующая информация:

- номера и длительности исходящих и входящих звонков;
- номера и размеры (символов) исходящих и входящих *SMS*-сообщений;
- номера и размеры (Кб) исходящих и входящих *MMS*-сообщений;
- тариф, определяющий стоимость:
- секунды исходящего и входящего звонка;
- исходящего *SMS*-сообщения;
- исходящего *MMS*-сообщения.

а) Разработать модуль, позволяющий редактировать базу данных.

б) Разработать модуль, позволяющий моделировать в течение заданного времени работу сети сотового оператора и автоматически заполнять базу данных. В качестве параметров задаются количество абонентов, а также одинаковые для каждого абонента вероятности исходящего и входящего звонков, отправки *SMS*- и *MMS*-сообщений.

в) Разработать модуль анализа базы данных сотового оператора с возможностью:

- получения всей статистики по выбранному номеру;
- общей статистики по базе данных, включая доход оператора;
- поиск самого «экономичного» абонента, самого «доходного» абонента;
- поиск тарифных показателей, позволяющих добиться максимального дохода при заданных параметрах моделирования работы сотового оператора.

6. **Игра «Тетрис».**

Разработать программу, которая выполняет перемещение фигур различной формы по экрану сверху вниз, их координацию в соответствии с действиями играющего (в автоматическом режиме, в режиме игры пользователя). Предусмотреть возможность выбора сложности формы фигур (уровень сложности).

а) Разработать модуль визуализации игры;

б) Разработать модуль оперирования фигурами (разворот, исчезновение линий удачно уложенных фигур), подсчет очков (в зависимости от уровня сложности и количества убранных одновременно линий);

в) Найти зависимость времени сеанса и игры в автоматическом режиме от уровня сложности.

7. **«Текстовый редактор».**

Разработать программу, реализующую основные функции текстового редактора.

а) Разработать модуль, предусматривающий функции редактирования текста, сохранения и загрузки текстового файла.

б) Разработать модуль, реализующий поиск/замену заданной подстроки, а также статистический анализ текста (количество строк, символов с пробелами, символов без пробела).

8. **«Интерпретатор языка».**

Программа поддерживает основные арифметические операции и унарный минус. Порядок операций задается скобками. Имена переменных задаются символом.

а) Разработать модуль редактирования и подсчета результата.

б) Разработать модуль анализа ошибок с выводом соответствующих сообщений.

1.2 Этапы выполнения работы

Общая длительность выполнения работы – 16 недель. В табл. 1 приведен график выполнения курсовой работы с ориентировочной длительностью каждого этапа. По истечении каждого этапа ожидаемый результат проходит промежуточное оценивание. Общая оценка выполнения работы определяется как сумма оценок всех этапов.

Таблица 1. График выполнения курсовой работы

№	Мероприятие	Срок, неделя	Ожидаемый результат	Стоимость
1	Формирование под-	1	утвержденное групповое (индиви-	5

	групп исполнителей, выбор задания		дуальное) задание, бланк – в приложении	
2	Изучение программных средств разработки, подготовка ТЗ	2	утвержденное ТЗ	12
3	Проектирование	4	• проект интерфейса пользователя; • структурная и/или функциональная схема ПО системы; • словесное описание каждого элемента схемы и существующих связей (информационных, функциональных).	15
4	Кодирование	10	код программы	25
5	Тестирование	12	план и результаты тестирования	13
6	Оформление ПЗ	13	ПЗ	20
7	Защита	15÷16	Слайды презентации	10
Всего:				100

Итоговая оценка за выполнение курсовой работы определяется следующим образом:

- «отлично» – 86 ÷ 100 баллов;
- «хорошо» – 71 ÷ 85 баллов;
- «удовлетворительно» – 50 ÷ 70 баллов.

2. Руководство к выполнению этапов

2.1 Техническое задание

Основным документом, в соответствии с которым выполняется разработка некоторого проекта в любой отрасли, включая проекты по разработке программного обеспечения (ПО), является техническое задание (ТЗ).

При разработке ПО техническое задание – технический документ (спецификация), оговаривающий перечень требований к системе и утверждённый как заказчиком/пользователем, так и исполнителем/производителем системы. Спецификация может содержать системные требования, требования к тестированию и др. Техническое задание позволяет обеим сторонам (заказчику и исполнителю) согласовать все необходимые детали реализации ПО, спланировать сроки и этапы выполнения проекта. Кроме того, ТЗ позволяет заказчику требовать от исполнителя соответствия продукта всем без исключения условиям, оговорённым в ТЗ, а исполнителю на законных основаниях отказаться от выполнения работ, не указанных в ТЗ.

Возможны различные варианты подготовки ТЗ:

- в соответствии с государственным стандартом ГОСТ 34.602-89, предполагающим детальное описание всех возможных аспектов разработки ПО и требующим подготовку значительного по объему документа;
- в соответствии со стандартом IEEE Std 830, предполагающим различные способы структурирования детальных требований для различных классов систем и позволяющим детализацию, достаточную для понимания;
- в соответствии с некоторым упрощенным корпоративным шаблоном оформления.

При разработке ТЗ в рамках выполнения курсовой работы предлагается использовать третий вариант подготовки ТЗ, предполагающего следующую структуру.

1. Введение
   - 1.1. Наименование продукта
   - 1.2. Краткая характеристика области применения
2. Основание для разработки
   - 2.1. Документ, на основании которого ведется разработка
   - 2.2. Организация, утвердившая документ
3. Назначение разработки
4. Требования к разработке
   - 4.1. Требования к функциональным характеристикам
   - 4.2. Требования к надежности
   - 4.3. Требования к составу и параметрам технических средств
   - 4.4. Требования к информационной и программной совместимости

5. Требования к программной документации
6. Технико-экономические показатели
7. Этапы разработки
8. Порядок контроля и приемки.

При подготовке ТЗ особое внимание следует уделить п.4.1. В нем необходимо детально указать перечень требований к разрабатываемому ПО так, чтобы не только полностью соответствовать заданию, но и расширить его за счет дополнительных функциональных характеристик.

При коллективном выполнении задания особое значение приобретает планирование работы и определение сферы ответственности каждого члена группы разработчиков. Соответствующая информация об этапах и сферах ответственности указывается в п.7. При этом следует избегать групповой ответственности за выполнение того или иного этапа, оставляя такую возможность только для исключительных случаев.

Пример, которым следует руководствоваться при подготовке ТЗ – в приложении.

2.2 Проектирование

Проектирование программного обеспечения является сложным процессом, который может выполняться как «вручную», так и с использованием развитых автоматизированных средств и различных нотаций – схем, *ER*-, *UML*-, *DFD*-диаграмм и др. Обычно при разработке ПО проектированию подлежит:

- архитектура (представление о компонентах системы, взаимосвязях между компонентами, а также правилах, регламентирующих эти взаимосвязи);
- устройство компонентов;
- пользовательский интерфейс.

Для простоты при выполнении курсовой работы реализуем проектирование архитектуры и пользовательского интерфейса без использования средств автоматизации проектирования.

2.2.1 Архитектура

При проектировании архитектуры ПО принимаются ключевые проектные решения относительно внутреннего устройства программной системы и её технических интерфейсов. При проектировании архитектуры в общем случае необходимо:

- определить базовую архитектурную парадигму (процедурно-ориентированная, объектно-ориентированная);
- разбить систему на подсистемы (слои, модули, компоненты);

• определить языковую парадигму для каждой подсистемы и выбрать средства разработки;

• разработать ключевые технические сценарии взаимодействия подсистем;

• определить протоколы взаимодействия компонентов (проектирование технических интерфейсов);

• определение форматов хранения и передачи данных;

• подобрать технические средства и шаблоны для реализации подсистем.

При выполнении курсовой работы реализуем проектирование архитектуры ПО системы путем разработки ее структурной и/или функциональной схемы ПО системы. На схеме должны быть однозначно отражены:

• все подсистемы (модули), планируемые к разработке;

• связи между подсистемами, с разделением их на управляющие и информационные;

• данные и операции над данными.

Пример обобщенной структуры некоторой информационной системы приведен на рис. 1.

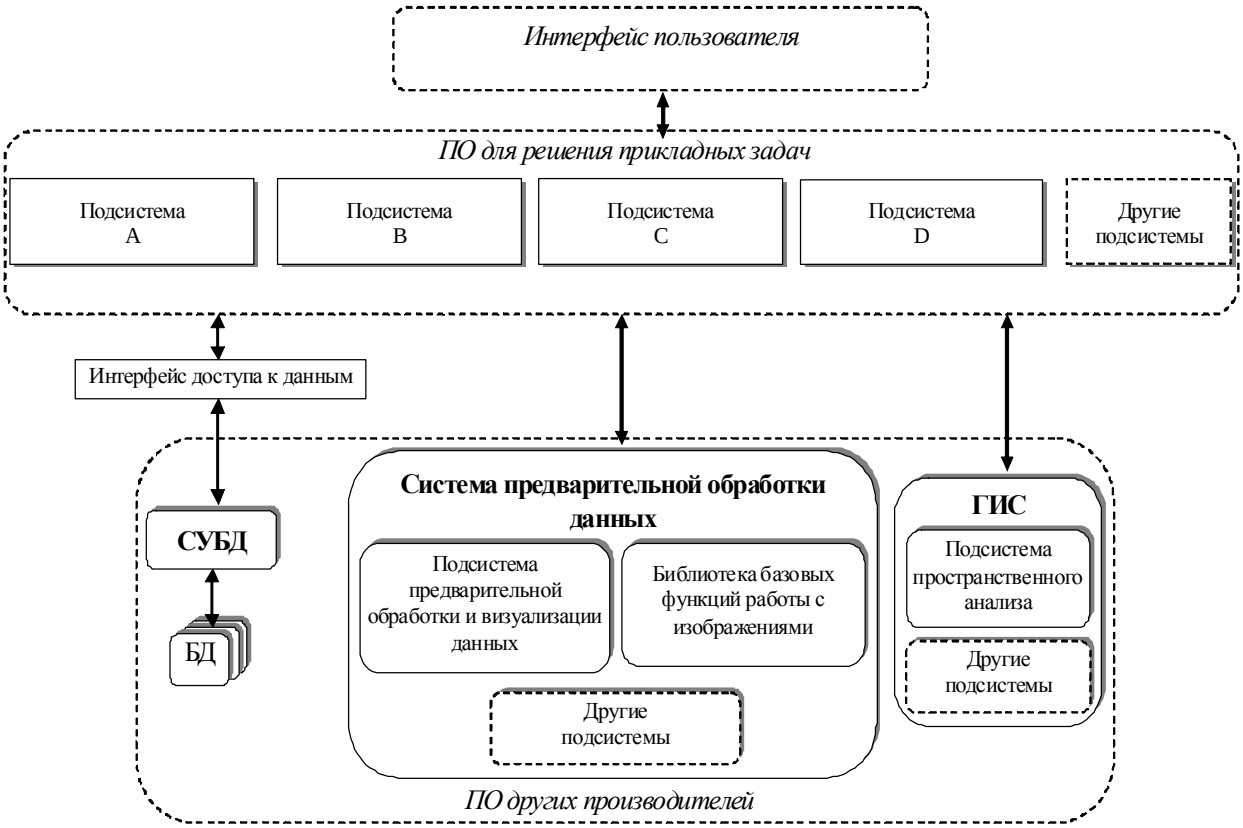

Рис. 1. Пример обобщенной структуры некоторой информационной системы

2.2.2 **Пользовательский интерфейс**

Пользовательский интерфейс имеет важное значение для любой программной системы и является неотъемлемой ее составляющей, ориентированной, прежде всего, на конечного пользователя. Именно через интерфейс пользователь судит о прикладной программе в целом. Более того, часто решение об использовании прикладной программы пользователь принимает по тому, насколько ему удобен и понятен пользовательский интерфейс. Вместе с тем, трудоемкость проектирования и разработки интерфейса может быть достаточно велика, и достигать более половины общего времени реализации проекта.

Основным предназначением системы является предоставление пользователю необходимой функциональности. Поэтому разработку интерфейса следует реализовать в следующей последовательности:

- определение перечня основных функций системы, которые должны быть отражены в интерфейсе (в данном случае в интерфейсе должны обязательно быть отражены позиции п.4.1 ТЗ);
- определение перечня окон, их предназначение и общее содержимое;
- определение диаграммы переходов между окнами;
- схематичное отображение детального содержимого каждого окна.

При разработке диаграммы переходов необходимо следовать, с точки зрения компромисса, двум противоречивым требованиям:

- диаграмма должна быть достаточно полной, чтобы из любой функции (если допустимо предметной областью) можно было бы перейти к любой другой функции (полный граф);
- диаграмма должна быть достаточно простой, не перегруженной множеством возможных переходов и избыточной информацией, непосредственно не требующейся в реализации той или иной функции.

Кроме того, при разработке интерфейса пользователя следует придерживаться следующих критериев качества:

1) *Удобство и интуитивность* **(**привычные названия, возможность самостоятельного изучения и использования функций системы, легкость работы с системой).

2) *Единообразие* **(**предпочтителен стандарт, принятый в операционной системе, недопустимо использование одинаковых функционально, но различных внешне элементов).

3) *Отсутствие перегруженности* **(**небольшое число объектов на экране – не более 10).

4) *Устойчивость* **(**по возможности предотвращение некорректных действий пользователя).

2.3 Кодирование

Набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования называется **стандартом оформления кода** или **стандартом кодирования** (англ. *coding standards*, *coding convention* или *programming style*).

Стандарт оформления кода обычно принимается и используется некоторой группой разработчиков программного обеспечения с целью единообразного оформления совместно используемого кода. Такой стандарт сильно зависит от используемого языка программирования. Например, стандарт оформления кода для языка *C/C++* будет серьёзно отличаться от стандарта для языка *BASIC*.

Обычно стандарт оформления кода описывает:

- способы выбора названий и используемый регистр символов для имён переменных и других идентификаторов (стиль именования переменных, констант и функций; запись типа переменной в её идентификаторе (венгерская нотация); регистр символов (нижний, верхний, «верблюжий», «верблюжий» с малой буквы), использование знаков подчёркивания для разделения слов);
- количество операторов в строке;
- стиль отступов при оформлении логических блоков – используются ли символы табуляции, ширина отступа; способ расстановки скобок, ограничивающих логические блоки;
- использование пробелов при оформлении логических и арифметических выражений; использование пустых скобок;
- стиль комментариев и использование документирующих комментариев;
- учет различных особенностей языка.

Стиль именования переменных, констант и функций. Соглашение об именах делает программы более понятными и упрощает их чтение. Также соглашение может дать информацию о функции, выполняемой тем или иным идентификатором. Например, является ли запись константой, пакетом или классом, что может быть полезным для понимания кода.

Классы и интерфейсы. Названия классов и интерфейсов должны быть именами существительными. Необходимо давать классам простые и понятные названия. Необходимо использовать целые слова без сокращений и аббревиатур (за исключением когда аббревиатуры являются общеизвестными, такие как *URL* или *HTML*).

```
class Raster;
class ImageSprite;
```

Методы. Названия методов должны быть глаголами, кратко описывающими суть выполняемых действий данного метода. В случае, если метод возвращает бинарное значение (например, логическое), имеет смысл в начале имени поставить глагол *is*.

```
Run();
RunFast();
GetBackground();
CheckModel();
isValidMove(5,2,5,4);
```

Переменные. Имена переменных, по возможности, должны быть короткими, но при этом отражать возложенные на переменную функции. Выбор имени переменной должен быть мнемонический, т.е. указывающий случайному читателю назначение той или иной переменной. Односимвольные имена являются исключением для временных «одноразовых» переменных. Обычно таким переменные дают следующие имена:

- *i*, *j*, *k*, *m*, *n* для целых типов;
- *c*, *d*, *s* для символьных типов.

Константы. Имена переменных, объявленных константами внутри класса, и ANSI констант должны содержать символы только в верхнем регистре с словами разделяемыми символом подчеркивания («_»). ANSI констант следует избегать для упрощения отладки.

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

Венгерская нотация. Суть венгерской нотации сводится к тому, что имена идентификаторов предваряются заранее оговорёнными префиксами, состоящими из одного или нескольких символов. При этом, как правило, ни само наличие префиксов, ни их написание не являются требованием языков программирования, и у каждого программиста (или коллектива программистов) они могут быть своими.

Применяемая система префиксов зависит от многих факторов:

- языка программирования (чем более «либеральный» синтаксис, тем больше контроля требуется со стороны программиста – а значит, тем более развита система префиксов. К тому же использование в каждом из языков программирования своей терминологии также вносит особенности в выбор префиксов);

• стиля программирования (объектно-ориентированный код может вообще не требовать префиксов, в то время как в «монолитном» для разборчивости они зачастую нужны);

• предметной области (например, префиксы могут применяться для записи единиц измерения);

• доступных средств автоматизации (генератор документации, навигация по коду, предиктивный ввод текста, автоматизированный рефакторинг и т.д.).

Примеры:

Префикс	Сокращение от	Смысл	Пример
s	string	строка	`sClientName`
sz	zero-terminated string	строка, ограниченная нулевым символом	`szClientName`
n, i	int	целочисленная переменная	`nSize, iSize`
l	long	длинное целое	`lAmount`
b	boolean	булева переменная	`bIsEmpty`
a	array	массив	`aDimensions`
t, dt	time, datetime	время, дата и время	`tDelivery, dtDelivery`
p	pointer	указатель	`pBox`
lp	long pointer	двойной (дальний) указатель	`lpBox`
r	reference	ссылка	`rBoxes`
h	handle	дескриптор	`hWindow`
m_	member	переменная-член	`m_sAddress`
g_	global	глобальная переменная	`g_nSpeed`
C	class	класс	`CString`
T	type	тип	`TObject`
I	interface	интерфейс	`IDispatch`

Как видно в приведенном примере, префикс может быть и составным. Например, для именования строковой переменной-члена класса использована комбинация префиксов «m_» и «s» (m_sAddress).

Верблюжий стиль и стиль через знак подчеркивания. CamelCase (русск. *ВерблюжийРегистр*, также *ГорбатыйРегистр*, *СтильВерблюда*) – стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово пишется с большой буквы. Стиль

получил название *CamelCase*, поскольку заглавные буквы внутри слова напоминают горбы верблюда (англ. *Camel*).

Различают два варианта *CamelCase*-написания в зависимости от того, прописная (большая) или строчная (маленькая) первая буква: *UpperCamelCase* (*PascalCase*) и *lowerCamelCase*.

Примеры *CamelCase*-написания: *BackColor*, *backColor*.

Принято считать альтернативным способом создания имён в программном коде *стиль_через_подчёркивание*. Хотя стили создания имён – всего лишь условная договорённость и никак не влияют на работу программы, тем не менее стили смешивать не рекомендуется (например: *среднее_КоличествоЯблок*), так как читаемость кода от этого не повышается, но сам программист рискует запутаться в методе присвоения имен переменным.

Стили появились из-за того, что в коде программы желательно иметь осмысленные имена переменных (описывающие смысл содержащегося в переменной значения), но не слишком длинные, а правила синтаксиса языков программирования налагают ограничения на средства для создания имён (к примеру, в большинстве языков допускается использование только буквенных символов, цифр и знака подчёркивания).

Количество операторов в строке. Для улучшения читаемости исходного текста программы рекомендуется писать не более одного оператора в строке, что вызвано особенностями человеческого восприятия текста. Кроме того, это облегчает пошаговую отладку в символьных отладчиках.

Не следует опасаться того, что программа слишком вырастет в длину, так как реальные программы и без того настольно длинны, что несколько "лишних" страниц (или даже десятков страниц) не меняют общую ситуацию. Выигрыш – удобочитаемость с избытком покрывает увеличение длины.

```
int  *ptr;  ptr  =  new  int  [100];
ptr[0] = 0;
```

Следует оформить следующим образом:

```
int *ptr;
ptr    = new int [100];
ptr[0] = 0;
```

В случае, когда требуется реализовать именно инициализацию переменной (об отличиях присвоения и инициализации переменных см. пособия), всё описание оформляется одной строкой:

```
int sum = 0;
```

Стиль отступов – правила форматирования исходного кода, в соответствии с которыми отступы проставляются в удобочитаемой манере. Редакторы текста, входящие в состав большинства популярных сред разработки, часто предоставляют средства для поддержки используемого стиля отступов, например, автоматическую вставку пробелов/табуляции при вводе скобок, обозначающих начало/конец логического блока.

Стиль «K&R». Назван в честь Кернигана и Ричи из-за того, что все примеры из их книги «*Язык программирования C*» (нередко обозначаемой как просто «*K&R*» по инициалам авторов) отформатированы подобным образом. Также известен как «*kernel style*» (из-за того, что ядро *UNIX* написано на нём), а также как «Единственный Правильный Скобочный Стиль» (англ. *One True Brace Style – 1TBS*) со слов его приверженцев. Основной отступ, показанный ниже, состоит из 8 пробелов (или одной табуляции) на уровень. Иногда (но реже) используются 4 пробела.

```
if (<cond>) {
........<body>
}
```

Стиль Олмана. Стиль Олмана – по имени Эрика Олмана, хакера из Беркли, написавшего множество *BSD*-утилит на нём (еще известен как «*стиль BSD*»). Имеет сходство с Паскалем и Алголом. Основной отступ на уровень – 8 пробелов, но не менее распространен стиль в 4 пробела (особенно в *C++*). Этот стиль по умолчанию предлагается в *Microsoft Visual Studio 2005* (и более ранних продуктах) и *Apple Xcode*.

```
if (<cond>)
{
........<body>
}
```

Стиль Уайтсмита. Популярен из-за примеров, шедших с *Whitesmiths C* – ранним компилятором с языка *C*. Основной отступ на уровень для скобок и блока — 8 пробелов.

```
if (<cond>)
........{
........<body>
........}
```

Стиль GNU. Используется во всех исходных текстах проекта *GNU* (например, *GNU Emacs*). Отступы всегда 4 символа на уровень, скобки находятся на половине отступа.

```
if (<cond>)
··{
····<body>
··}
```

Исследования показали наибольшее распространение стилей Олмана и Уайтсмита, с примерно равным количеством поклонников на каждый. *K&R/1TBS* считается более универсальным, но распространен сейчас мало (открывающая скобка имеет тенденцию теряться). Защитники *1TBS* приводят в защиту стиля экономность в использовании вертикального пространства, когда на одном экране можно прочесть большее количество строк текста.

По мнению авторов пособия стиль *GNU*, несомненно, хуже остальных, поскольку не только не привносит какой-либо дополнительной ясности, но и несколько снижает ее. Во-первых, выбор положения скобок где-то внутри отступа произволен и ничем не обоснован. Во-вторых, количество вертикальных *зрительных линий отступов* из-за этого удваивается и составляет вместо обычных четырех-пяти линий восемь-десять, что снижает наглядность без повышения информативности.

Пробелы должны использоваться в следующих случаях:

- Ключевое слово и следующая за ним открывающая скобка должны быть разделены пробелом. Например:

```
while (true) {
        ...
}
```

Пробелы не должны разделять название метода и следующую за ним открывающую скобку. Это помогает отличать ключевые слова от вызова методов.

- Пробел должен стоять после запятой в списке аргументов.
- Все бинарные операторы исключая . должны быть разделены при помощи пробела. Пробел никогда не разделяет унарные операторы такие как унарный минус, инкремент (“++”),и декремент (“--”) от их операндов. Например:

```
a += c + d;
a = (a + b) / (c * d);
while (d++ = s++) {
    n++;
}
prints("size is " + foo + "\n");
```

- Выражения в цикле *for* должны быть разделены пробелами. Например:

```
for (expr1; expr2; expr3)
```

Пустые строки. Использование пустых строк является важным средством для выделения участков программы. При этом имеет смысл отделять:

1) определения переменных:

```
char str[80];
int  counter = 0;

fgets( str, 79, infile);
counter++;
```

2) последовательности однотипных инструкций или директив:

```
#include
#include
#include

#define NAME_SIZE 256
#define MAX_LEN 3000
```

3) функции:

```
int main()
{
   . . .
}

char *get_name(FILE *f)
{
   . . .
}
```

4) любые логически завершенные блоки кода:

```
printf( "Enter size and delta: " );    //Блок ввода данных
scanf( "%d", &size );
scanf( "%f", &delta );

for( i=0; i < size; i++ )              //Блок использования данных
{
  a[i] -= delta;
  b[i] += delta;
}
```

Комментарии. Большинство специалистов сходятся во мнении, что комментарии должны объяснять намерения программиста, а не код; то, что можно выразить на языке программирования, не должно выноситься в комментарии – в частности, надо использовать говорящие названия переменных, функций, классов, методов и пр., разбивать программу на лёгкие для понимания части, стремиться к тому, чтобы структура классов и структура баз данных были максимально понятными и прозрачными и т. д. Есть даже мнение (его придерживаются в экстремальном программировании и некоторых других гибких методологиях программирования), что если для понимания программы требуются комментарии — значит, она плохо написана.

Концепция грамотного программирования настаивает на включение в текст программы настолько подробных и продуманных комментариев, чтобы она стала исходным текстом не только для исполняемого кода, но и для сопроводительной документации.

Время, потраченное на написание комментариев, многократно окупится при любых модификациях программы. Однако комментировать все подряд, включая самоочевидные действия, как в следующем примере, тоже не стоит:

```
size = 10;                      // Присвоить size значение 10
for( i=0; i < size; i++)        // Цикл по i от 0 до size
{
   . . .
}
```

Комментировать следует:

- заголовок файла, описывая содержимое данного файла;
- заголовок функции, поясняя назначение ее аргументов и смысл самой функции;
- вводимые переменные и структуры данных;
- основные этапы и особенности реализуемых алгоритмов;
- любые места, которые трудны для быстрого понимания, в особенности использование различных программных "трюков" и нестандартных приемов.

Некоторые комментарии программисты используют в ходе своей работы. Подобные комментарии особенно полезны, когда над одним кодом работает несколько разработчиков. Так, комментарием *TODO* обычно помечают участок кода, который программист оставляет незавершённым, чтобы вернуться к нему позже. Комментарий *FIXME* помечает обнаруженную ошибку, которую решают исправить позже. Комментарий *XXX* или *ZZZ* обозначает найденную критическую ошибку, без исправления которой нельзя продолжать дальнейшую работу.

Учет различных особенностей языка.

1. *Скобки.* Необходимо использовать скобки при описании арифметических операций для избежания проблем с их обработкой. Даже если оператор кажется понятен, необходимо помнить, что он может быть непонятен другим программистам, которые будут работать с кодом.

```
if (a == b && c == d)           // Необходимо избегать!
if ((a == b) && (c == d))       // Следует использовать
```

2. *Возвращаемые значения.* Старайтесь сделать структуру программы компактнее. Например:

```
if ( booleanExpression) {
     return true;
} else {
     return false;
}
```

```
if ( условие) {
     return x;
}
return y;
```

Необходимо писать:

```
return booleanExpression;
```

```
return (condition ? x : y);
```

3. *Присвоение значений переменным*. Необходимо избегать присвоения значения нескольким переменным одного значения в одном выражении, т.к. это затрудняет чтение кода.

```
fooBar.fChar = barFoo.lchar = 'c';      // Следует избегать!
d = (a = b + c) + r;                    // Следует избегать!
```

Второй случай рекомендуется изменить на следующую запись:

```
a = b + c;
d = a + r;
```

4. *Присвоение и сравнение*. Не используйте операторы присвоения в местах, где они могут быть спутаны с оператором сравнения. Например:

```
if (c++ = d++) { // Следует избегать!
    ...
}
```

Должно быть написано:

```
if ((c++ = d++) != 0) {
    ...
}
```

2.4 Тестирование

Вероятно, одной из самых больших трудностей при разработке качественного ПО является обеспечение целостности и согласованности всех действий и требуемых результатов, в особенности при многочисленной команде разработчиков. Компании-производители коммерческого ПО стремятся повысить качество программных продуктов с помощью тестирования. Существуют специальные драйверы, автоматизирующие процесс тестирования раз-

рабатываемого ПО. Также используется «*бета-тестирование*», при котором разработчики передают пользователям пробные предварительные версии разрабатываемых систем. При этом даже после распространения финальных версий своих программных продуктов производители коммерческого ПО продолжают искать и исправлять ошибки, выпуская «*пакеты обновлений*» и «патчи».

Таким образом, тестирование – один из основных инструментов обеспечения безотказной корректной работы ПО, в конечном итоге влияющим на общее качество и коммерческую конкурентоспособность программного продукта. В практике программирования наиболее часто в роли метрики качества продукта выступает остаточная плотность ошибок, то есть плотность ошибок на тысячу строк кода или на одну функциональную точку.

Тестирование ПО – процесс поиска ошибок, заключающийся в выявлении отличий ожидаемых результатов работ ПО от фактических. Несмотря на разнообразие существующих подходов к тестированию ПО, в том числе с использованием средств автоматизации, следует признать, что тестирование сложных программных систем – это процесс в значительной степени творческий, не сводящийся к следованию строгим и чётким процедурам. При этом очевидно, что тестирование не позволяет полностью избавиться от ошибок в ПО, а лишь может позволить (при правильном планировании и добросовестном выполнении) существенно уменьшить их количество.

В общем виде тестирование предусматривает последовательное выполнение следующих этапов:

- разработку плана тестирования;
- разработку тестовых заданий;
- выполнение тестовых процедур;
- формирование заключения по результатам.

План тестирования должен содержать:

- описание объекта тестирования (система, клиентское приложение, оборудование) и тестовой среды (например, операционная система клиентского приложения);
- критерии начала тестирования (готовность тестовой платформы, законченность разработки требуемого функционала, наличие необходимой документации);
- критерии окончания тестирования (результаты тестирования удовлетворяют критериям качества продукта, выдержка определенного периода без изменения исходного кода приложения, выдержка определенного периода без появления новых ошибок);
- виды тестирования и их применение к тестируемому объекту (например, тестирование основных сценариев, тестирование с некорректными

действиями пользователя, нагрузочное тестирование, тестирование аварийных ситуаций и т.п.);

- последовательность тестирования (подготовка, тестирование, анализ результатов);
- спецификацию тестирования (список функций и/или компонент тестируемой системы).

После подготовки плана тестирования разрабатывают *тестовые задания* (*Test Cases*) – совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Тестовое задание может иметь структуру вида *<действие>* → *<ожидаемый результат>* → *<фактический результат>*.

Очевидно, что возможны различные уровни детализации при разработке тестовых заданий. Целесообразно использовать такую детализацию, которая позволяет достичь разумного соотношения времени выполнения тестового задания к «тестовому покрытию».

Таблица 2. Примеры тестовых заданий

Вариант использования	Действие	Ожидаемый результат	Фактический результат
Проверка корректности отображения окна	Открыть окно «Вход в систему»	• окно «Вход в систему» открыто; Название окна – «Вход в систему» Логотип компании отображается в правом верхнем углу; • на форме 2 поля – Имя и Пароль; • кнопка «Вход» доступна; Ссылка «забыл пароль» – доступна.	–‖–
	Закрыть страницу «Вход в систему»	Окно корректно закрывается	Ошибка «Access violation…»

После выполнения запланированных тестовых процедур следует подготовить заключение о результатах тестирования, позволяющее сделать вывод об устойчивости и корректности работы при различных условиях (видах тес-

тирования, тестовых заданиях) отдельных модулей и подсистем, а также системы в целом.

Основным требованием к такому заключению является то, что при внешней оценке оно должно позволить сделать вывод либо об успешном завершении этапа тестирования и возможности передачи разработанной системы в опытную эксплуатации, либо о необходимости ее доработки (с указанием – в какой части: подсистема, возможные причины и пути устранения).

2.5 Защита

Заключительным этапом работы является ее защита. Умение кратко и ясно для аудитории изложить основные результаты работы является очень важным, часто даже не менее важным, чем собственно практические результаты работы и их оформление в виде соответствующей пояснительной записки. Ведь часто исключительно по результатам защиты и презентации могут судить о результатах работы в целом.

Защита осуществляется публично с использованием презентации (например, выполненной в MS PowerPoint) общим объемом 10-12 слайдов. Рекомендуемое содержание презентации следующее:

- Задание;
- Функциональные характеристики (согласно ТЗ п. 4.1);
- Проект;
- Особенности программной реализации;
- Демонстрация работоспособности;
- Тестирование;
- Выводы.

Регламент выступлений:

- доклад 5-6 мин. (2-3 мин. каждый участник подгруппы);
- 2-3 мин. – вопросы, дискуссия.

3. Правила оформления текстовой части

3.1 Требования к оформлению пояснительной записки

При оформлении пояснительной записки стоит обращать внимание на количество используемых стилей (в редакторе MS Word: меню Формат→Стили и форматирование) – в списке стилей должны быть только необходимые (остальные следует скрыть или удалить), это поможет привести части документа к единообразию.

1. Используются поля следующих размеров: слева – 25 мм; справа – 10 мм; сверху – 15 мм; снизу – 20 мм.
2. Нумерация страниц – сквозная, номер проставляется в правом верхнем углу страницы. На титульном листе номер не ставится.
3. Заголовки разделов и подразделов должны быть сформулированы кратко, на первом месте должно стоять имя существительное.
4. Все заголовки иерархически нумеруются. В конце заголовка точка не ставится. Такие разделы как «СОДЕРЖАНИЕ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСТОЧНИКОВ», «ПРИЛОЖЕНИЕ» не нумеруются.
5. Каждый раздел (заголовок 1-го уровня) следует начинать с новой страницы. Заголовок 1-го уровня следует располагать в середине строки и набирать прописными буквами. Заголовки 2-го уровня и ниже следует начинать с абзацного отступа и печатать с прописной буквы.
6. Заголовки следует отделять от окружающего текста промежутком размером не менее чем в 15 мм снизу и 30 мм сверху. После любого заголовка должен следовать текст, а не рисунок, формула или таблица.
7. Основной текст рекомендуется набирать шрифтом семейства *Times* с обычным начертанием. Заголовки 1-го и 2-го уровней набираются шрифтом семейства *Arial* с полужирным начертанием, заголовки 3-го уровня – Times полужирным, заголовки 4-го уровня – *Times* обычным. Названия рисунков и таблиц следует набирать шрифтом *Times New Roman* полужирным, размером 12 пунктов. Размер шрифта для текста и заголовков – 14 пунктов. Абзацный отступ – 8 мм.
8. Исходный текст программ оформляется шрифтом *Courier New*, без абзацного отступа, с одинарным межстрочным интервалом, размером 12 пунктов или менее.
9. Все рисунки, таблицы и формулы нумеруются. Нумерация может быть либо сквозной по всему тексту, например «таблица 7», либо по разделам, например «Рисунок – 2.5», что означает рисунок 5 в разделе 2. Ссылаться на рисунок следует как «рис. 2.5», на таблицу – «табл. 7». Номер формулы располагается справа от нее в круглых скобках. Нумеруются только те формулы, на которые есть ссылки в тексте.

10. Каждый рисунок должен иметь название. Название для таблиц не обязательно. Точка после названия рисунка и таблицы не ставится.

11. Название рисунка располагается под рисунком по центру. Название таблицы располагается над таблицей справа.

12. На каждый рисунок, таблицу и приложение в тексте должна быть ссылка в основной части пояснительной записки.

13. Для каждого вида текста (обычный, заголовки различных уровней, подписи к рисункам и таблицам) рекомендуется создать соответствующий стиль в текстовом редакторе и использовать его.

14. В разделе «СПИСОК ИСТОЧНИКОВ» помещаются только те источники, которые использовались при написании текста.

15. Ссылки на литературные источники в тексте приводятся в квадратных скобках, например, [1, 2] или [1, 3–7].

16. Приложения идентифицируются номерами или буквами, например «ПРИЛОЖЕНИЕ 1» или «ПРИЛОЖЕНИЕ А». На следующей строке, при необходимости, помещается название приложения, например «ЛИСТИНГ ПРОГРАММЫ», которое оформляется как заголовок 1-го уровня без нумерации.

3.2 Требования к руководству пользователя

Одной из важнейших составляющих любой законченной программы является руководство пользователя. От того, насколько понятно и доступно написано руководство, зависит успех программы, ее распространенность и популярность.

Существует 3 основных типа руководств:

1. Описание с пошаговой инструкцией. Это наиболее распространённый тип, в котором даётся руководство по выполнению различных действий в программе с точки зрения целей пользователя: каждое действие разбивается на несколько простых операций и пользователю предъявляется информация о том, каким образом и в какой последовательности он может их выполнять (как правило, описание сопровождается большим количеством изображений кнопок, панелей, форм и т.д.).

2. Другим типом является описание, ориентированное на перечисление и описание операций, допустимых в программе. В этом случае структура руководства будет совпадать с иерархией программных операций. В этом типе руководства не описывается, каким образом можно решить какую-либо довольно крупную задачу, предоставляя пользователю самому размышлять над этим вопросом.

3. И, наконец, последним типом является руководство для начинающих, которое помогает новичкам ознакомиться с программным продуктом, узнать, как работает программа и как управлять интерфейсом.

Несмотря на довольно существенные различия в подходах к написанию руководства пользователя, можно рекомендовать следующие обязательные разделы:

1. «О программе» – описание программы, назначение и основные возможности.
2. «Системные требования» – список аппаратных и программных средств и их характеристик, необходимых для запуска и успешного функционирования программы.
3. «Интерфейс» – описание интерфейса программы, основных элементов управления и горячих клавиш. При наличии графического интерфейса необходимо привести скриншоты, иллюстрирующие интерфейс пользователя.
4. «Запуск программы» – описание действий, необходимых для запуска программы.
5. «Работа с программой» – пошаговое описание основных действий (в соответствии с выбранным подходом к написанию руководства), которые доступны в программе, с пояснениями и скриншотами-примерами.
6. «Приложение». Необязательный раздел, добавляется при необходимости и может включать любые сведения, не вошедшие в вышеперечисленные разделы, например, глоссарий.

Необходимо учитывать, что руководство пользователя пишется для людей, которые, вполне возможно, плохо умеют работать с компьютером и не знакомы со многими понятиями информационных технологий. Поэтому руководство пользователя следует писать понятным языком, по возможности употребляя как можно меньше специфической терминологии и аббревиатур. Также не допускается использование жаргонной лексики.

Структура предложений должна быть как можно более простой, не перегруженной сложными речевыми оборотами.

4. Учебно-методическое обеспечение

4.1 Литература обязательная

1. Триханова Н.В. Программирование на языке С++. Часть 1: Учебное пособие / Н.В. Триханова. — Томск: Изд. ТПУ, 1999. — 96с.

2. Алгоритмические языки и программирование. Часть 2. Программирование на языке С++: Рабочая программа дисциплины, методические указания и контрольная работа для студентов специальности 220100 “Электронные вычислительные машины, системы, комплексы и сети” ЦДО / Н.В. Триханова, И.Г. Смышляева. — Томск: Изд. ТПУ, 2000. — 26 с.

3. Программирование на языке С++. Часть II. Учебное пособие / Н.В. Триханова, Е.А. Мирошниченко. — Томск: Изд. ТПУ, 2001. — 118с.

4. Турбо С++: язык и применение/ А.А. Цимбалл, А.Г. Майоров, М.А.Козодоев. — М.: Джей АЙ Лтд, 1993.

5. Программирование на С++/ С. Дьюхарст, К. Старк. — Киев: НИПО “ДиаСофт”, 1993.

6. Вайнер Р. и др. С++ изнутри/ Р. Вайнер, Л. Пиксон. — Киев: НИПО “ДиаСофт”, 1993.

7. Буч Г. Объектно-ориентированное программирование с примерами применения/ Г. Буч. — Киев: НИПО “ДиаСофт”, 1993.

8. От Turbo C к Borland C++: Справочное пособие. / А.И. Касаткин, А.Н. Вальвачев. — Минск: Вышейшая школа, 1992.

9. Подбельский А.А. Язык С++: Учебное пособие/ А.А.Подбельский. — М.: Финансы и статистика, 1995г. — 556с.

10. Програмирование на С++: учебное пособие./ В.П. Аверкин, А.И. Бобровский, В.В. Веснич, В.Ф. Радушинский, А.Д. Хомоненко. Под ред. А.Д. Хомоненко — СПб.: КОРОНА принт, 1999.-256с.

11. Программирование на С и С++. Практикум: Учебное пособие для вузов/ А.В.Крячков, И.В.Сухинина, В.К. Томшин. — М.: Радио и связь, 1997. — 344с.

12. Бабэ Б. Просто и ясно о Borland C++/ Б.Бабэ. — СПб.: Питер, 1997. — 464с.

13. Бад Т. Объектно-ориентированное программирование в действии:/ Т. Бад — СПб.: Питер, 1997. — 464с.

14. Ирэ П. Объектно-ориентированное программирование с использованием С++ /П.Ирэ .-К.: НИПФ ДиаСофтЛтд, 1995. — 480с.

15. Свердлик А.Н. Программирование. Учебник./ А.Н. Свердлик. — МО СССР, 1992. — 608с.

16. Костюк Ю.Л. Основы алгоритмизации: Учебное пособие. / Ю.Л. Костюк. — Томск: Изд. ТГУ, 1996. — 124с.

17. Основы программирования. / В.М. Бондарев, В.И. Рублинецкий, Е.Г.

Качко / Худож.-оформитель С.А. Пяткова. — Харьков: Фолио; Ростов н/Д: Феникс, 1997. — 368с.

18. Вирт Н. Алгоритмы и структуры данных / Н. Вирт. — М.: Мир, 1989. — 310с.

19. Алгоритмы, построение и анализ / Т. Кормен, Ч. Лейзерсон. — М.: Р. Ривест МЦНМО, 2001. — 980с.

20. Кнут Д. Искусство программирования для ЭВМ. В 3-х томах./ Д. Кнут. — М.: Наука, 1999.

21. Структуры данных для персональных ЭВМ./ Й.Лэнгсам, М.Генстайн. — М.: Мир, 1989.

22. Справочник по алгоритмам и программам на языке бейсик для персональных ЭВМ/ Дьяконов В.П. — М.: Наука, Гл. ред. Физ.-мат. Лит., 1987. — 240 с.

23. Дейтел Х. Как программировать на С++/ Пер. с англ./Леен Аммераль. — М.: ЗАО “Издательство БИНОМ”, 1998. — 1024 с.

24. Вычислительные методы для инженеров / А.А. Амосов и др.. — М.: Высшая школа, 1994.

25. Демидович Б.П. и др. Основы вычислительной математики.— М.: Наука, 1970.

26. Крылов А.Н. Лекции о приближенных вычислениях. — М.: ГИФМЛ, 1950.

27. Крылов В.И. Приближенное вычисление интегралов. — М.:ГИФМЛ, 1959.

28. Машинные методы математических вычисления/ Дж. Формайт и др. — М.: Мир, 1980.

29. Хемминг Р.В. Численные методы. — М.: Наука, 1972.

30. Диберти Д. С++ энциклопедия пользователя/ Пер. с англ. Джесс Диберти. — М.: Диа-Софт, 1999.

31. Грис Д. Наука программирования: Пер. с англ. – М.: Мир, 1984. – 416 с, ил.

4.2 Литература дополнительная

32. Лукас П.С. С++ под рукой / П.С. Лукас. — Киев НИПО “ДиаСофт”, 1993.

33. Сван Т. Программирование для Windows в Borland С++/ Т.Сван. — М.: БИНОМ. — 480с.

34. Шамис В.А. Borland С++ Builder. Программирование на С++ без проблем / В.А. Шамис. — М.: Нолидж, 1997. — 266с.

35. Шилдт Г. Теория и практика С++/ Г. Шилдт. — СПб.: BHV — Санкт-Петербург, 1996. — 416с.

36. Шилдт Г. Самоучитель С++, 3-е издание/ Г. Шилдт. — СПб.: BHV — Санкт-Петербург, 1998. — 688с.

4.3 Учебно-методические пособия

37. Электронный учебник по программированию на языках высокого уровня.– Режим доступа: http://ad.cctpu.edu.ru/cpp/default.htm, вход свободный.

38. Интернет-курс по программированию на C++. http://www.intuit.ru/ department/ pl/ cpp/

39. Интернет-курс по объектно-ориентированному программированию. http://www.intuit.ru/department/se/oopbases/

4.4 Интернет-ресурсы

40. Официальный сайт Комитета стандартов C++. http://www.open-std.org/jtc1/sc22/wg21/

41. Сеть ресурсов по C++ (The C++ Resources Network). http://www. cplus-plus. com/

42. Домашняя страница Бьярна Страуструпа, создателя C++. http://www. research. att. com/ ~bs/

43. Часто задаваемые вопросы по C/C++ в группе на alt.comp.lang.learn.c-c++. http://www.faqs.org/faqs/C-faq/learn/

44. Библиотеки функций для C++. http://www.trumphurst.com/cpplibs/ cpplibs.php

45. Обзор языков высокого уровня. http://pmi.ulstu.ru/new_project/ hi_level_lang/

46. Сайт о программировании на языках высокого уровня. http://coding. tomsk. ru

Приложения

Кафедра ЕНДиИТ

УТВЕРЖДАЮ

Зав. кафедрой_____

ЗАДАНИЕ
на выполнение курсовой работы по дисциплине
“Программирование на языке высокого уровня”

Студенту(ам) гр._____
_____

1. Тема курсовой работы_____
_____

2. Срок сдачи работы: _____
3. Исходные данные к работе:
Язык программирования C++, компилятор Borland C++ Builder.
4. Содержание пояснительной записки
- Техническое задание;
- Введение;
- Проект;
- *Описание алгоритма, особенности реализации*;
- Тестирование;
- Заключение;
- Список использованной литературы;
- Приложение (руководство пользователя)

5. Дата выдачи задания: _____

Руководитель_____
(подпись, дата)

Задание принял к исполнению

_____
(подпись, дата)

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
на разработку графического редактора

1. ВВЕДЕНИЕ

1.1. Наименование продукта

Графический редактор со стандартным набором функций «Граф».

1.2. Краткая характеристика области применения

Предназначен для создания несложных рисунков с использованием графических примитивов.

2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

2.1. Документ, на основании которого ведется разработка

Задание на курсовую работу по дисциплине «Программирование на языке высокого уровня».

2.2. Организация, утвердившая документ

Государственное образовательное учреждение высшего профессионального образования «Томский политехнический университет».

3. НАЗНАЧЕНИЕ РАЗРАБОТКИ

Программа, с помощью которой можно будет создавать рисунки, сохранять их на диске и открывать снова для редактирования.

4. ТРЕБОВАНИЯ К РАЗРАБОТКЕ

4.1. Требования к функциональным характеристикам

4.1.1.Управление интерфейсом осуществляется с помощью курсора манипулятора «мышь».

4.1.2. Меню со стандартным набором геометрических фигур (квадрат, круг, прямоугольник, эллипс, линия, многоугольник).

4.1.3. Возможность вставки текста в рисунок.

4.1.4. Возможность изменения цвета рисования.

4.1.5. Наличие ластика, позволяющего стирать отдельные фрагменты рисунка.

4.1.6. Возможность полностью очищать экран для рисования.

4.1.7. Возможность сохранения рисунка в формате .bmp

4.2. Требования к надежности

4.2.1. Безотказная работа программного обеспечения редактора «Граф» при условии безотказной работы операционной системы ЭВМ.

4.3. Требования к информационной и программной совместимости

Для функционирования данного графического редактора необходимы операционная система Windows XP/Vista.

5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Состав программной документации:

- Техническое задание;
- Пояснительная записка;
- Руководство пользователя.

6. ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Графический редактор – аналог известных Paint, Photoshop.

7. СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

Техническое задание (ТЗ), пояснительная записка (ПЗ), программное обеспечение (ПО).

№	Содержание этапа	Срок		Форма отчетности	Ответственный
		начало	конец		
1.	Выбор и утверждение задания			ПЗ	Иванова А.В., Петров Н.А.
2.	Разработка и утверждение технического задания			ТЗ	Иванова А.В.

№	Содержание этапа	Срок		Форма отчетности	Ответственный
		начало	конец		
3.	Проектирование программы				
	• структурная схема, функциональная схема			ПЗ	Иванова А.В.
	• спецификация основных функций				Петров Н.А.
4.	Кодирование программы				
	• меню, палитра, выбор цвета			исходный код	Иванова А.В.
	• управление мышью				Петров Н.А.
	• круг, квадрат, прямоугольник				Петров Н.А.
	• линия, текст, заливка цветом				Петров Н.А.
5.	Тестирование программы				
	• тестирование основных сценариев			ПЗ	Иванова А.В.
	• тестирование некорректных действий				Петров Н.А.
6.	Оформление пояснительной записки				
	• разделы 1-4			ПЗ	Петров Н.А.
	• разделы 5-8				Иванова А.В.
7.	Защита курсовой работы				
	• слайды 1-8			слайды	Иванова А.В.
	• слайды 9-12, демонстрация работоспособности ПО				Петров Н.А.

8. ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

Приемка осуществляется в соответствии с Техническим заданием.