УНИВЕРСИТЕТ ИТМО

В.Ю. Петров

ИНФОРМАТИКА
АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Учебное пособие

Санкт-Петербург

2016

Петров Вадим Юрьевич. Информатика. Алгоритмизация и программирование. Учебное пособие. Часть 1. – СПб: Университет ИТМО, 2016. – 91с.

Учебное пособие представляет собой вводный курс по построению алгоритмов и программированию на универсальном языке программирования C++. В нем уделено внимание и компиляторам Borland C++ и VC++. Рассмотрены отличия при использовании первого и второго. Подробно рассмотрены теоретические аспекты построения алгоритмов и основ программирования. В пособии уделено место вопросам оформления выпускных квалификационных работ, рефератов и диссертаций. Изложение материала сопровождается большим количеством примеров.

Для студентов экономических специальностей 38.03.01. «Экономика предприятий и организаций», «Макроэкономическое планирование и прогнозирование»; 38.03.02. «Производственный менеджмент», «Управление проектом», «Финансовый менеджмент».

Рекомендовано к печати на заседании ученого совета факультета технологического менеджмента и инноваций, протокол № 5 от 22.12..15.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

ОГЛАВЛЕНИЕ

1. ОФОРМЛЕНИЕ ОТЧЕТОВ, РЕФЕРАТОВ, ВЫПУСКНЫХ КВАЛИФИКАЦИОННЫХ РАБОТ

1.1. РАЗНОВИДНОСТЬ ДОКУМЕНТОВ

Оформление любой научной работы - отчет по практической или лабораторной работе, реферат, дипломная работа, выпускная квалификационная работа, диссертация, научно-исследовательская работа – требует соблюдения определенных правил. Эти правила содержатся в государственных стандартах – ГОСТах, требованиях единой системы конструкторской документации – ЕСКД и требованиях единой системы программной документации - ЕСПД. Успех выполнения и оформления работы, ее оценки и защиты во многом зависит от соблюдения этих правил.

Коротко рассмотрим, что представляют собой реферат, диссертация, выпускная квалификационная работа, научно-исследовательская работа.

1.1.1. Реферат

Слово *«реферат»* имеет латинские корни и происходит от слова *«referre»*, что в буквальном переводе означает «докладывать, сообщать» [10]. Реферат имеет несколько значений (согласно словарю иностранных слов, изд. «Русский язык», Москва, 1986г.). Первое из них трактует слово «реферат» как доклад на определенную тему, включающий обзор соответствующих литературных и других источников.

Виды рефератов

Продуктивный реферат - содержит творческое или критическое осмысление реферируемого источника

Реферат-обзор составляется на основе нескольких источников и сопоставляет различные точки зрения по данному вопросу.

Реферат-доклад, наряду с анализом информации первоисточника, есть объективная оценка проблемы; этот реферат имеет развёрнутый характер.

Репродуктивный реферат - воспроизводит содержание первичного текста

Реферат-резюме содержит только основные положения данной темы.

Реферат-конспект содержит фактическую информацию в обобщённом виде, иллюстрированный материал, различные сведения о методах исследования, результатах исследования и возможностях их применения

Целью реферативной работы является приобретение навыков работы с литературой, обобщения литературных источников и практического материала по выбранной теме, способности грамотно излагать вопросы темы и делать выводы.

Общий объём работы - 15—30 страниц печатного текста (с учётом титульного листа, содержания и списка литературы) на бумаге формата А4, на одной стороне листа.

1.1.2. Выпускная квалификационная работа, дипломная работа (проект)

Дипломная работа (проект) — один из видов *выпускной квалификационной работы* — самостоятельная творческая работа студентов, обучающихся по программам подготовки специалистов или бакалавров, выполняемая ими на последнем, выпускном курсе [10].

Дипломная работа обычно выполняется студентами, обучающимися по естественно-научным, гуманитарным, экономическим и творческим специальностям, и имеет целью систематизацию, обобщение и проверку специальных теоретических знаний и практических навыков выпускников.

Дипломный проект, как правило, выполняется студентами, обучающимися по техническим специальностям, и предполагает создание или расчёт некоторого технического устройства или технологии. В виде исключения студенты технических вузов могут выполнять и дипломную работу, если она носит теоретический или экспериментальный характер.

Подготовка дипломного проекта (работы) и предшествующая этому профессионально-ориентированная практика, как заключительный этап обучения, отвечают за формирование у студентов навыков самостоятельной работы в профессиональной области.

Основной целью написания дипломной работы является систематизация и обобщение теоретических знаний и практических навыков выпускников. Дипломная работа, как правило, состоит из теоретической, практической (аналитической) и проектной части. Выполняется в большинстве случаев на примере конкретного предприятия.

Общий объём работы – 70-90 страниц печатного текста на бумаге формата А4 на одной стороне листа (с учётом титульного листа, содержания и списка литературы).

1.1.3. Диссертация

Диссертация (от лат. *dissertatio*— сочинение, рассуждение, доклад) — квалификационная работа на присуждение *академической* или *ученой степени* и *квалификации (степени) магистра* [7, 10].

В России различают диссертации на соискание учёной степени кандидата наук и доктора наук. В других странах существуют диссертации на соискание учёной степени доктора наук, доктора философии.

Требования к содержанию диссертации различаются в зависимости от учёной степени, на которую претендует соискатель, и от научного направления. Общими требованиями являются оригинальность, научная новизна и практическая значимость работы. Помимо собственно диссертации, от соискателя требуется наличие официально опубликованных печатных работ по теме диссертации. Диссертация принимается в ходе процедуры, называемой *защитой диссертации*. Для соискателей степени кандидата наук устанавливаются также дополнительные испытания в виде предшествующих защите кандидатских экзаменов.

Общий объём работы – 70-110 страниц печатного текста (с учётом титульного листа, содержания и списка литературы) на бумаге формата А4, на одной стороне листа.

1.1.4. Отчет о научно-исследовательской работе - НИР

Отчет о НИР — научно-технический документ, который содержит систематизированные данные о научно-исследовательской работе, описывает состояние научно-технической проблемы, процесс и/или результаты научного исследования [1].

По результатам выполнения НИР составляется заключительный отчет о работе в целом. Кроме того, по отдельным этапам НИР могут быть составлены промежуточные отчеты, что отражается в Техническом задании на НИР и в календарном плане выполнения НИР.

Ответственность за достоверность данных, содержащихся в отчете, и за соответствие его требованиям настоящего стандарта несет организация-исполнитель.

Отчет о НИР подлежит обязательному нормоконтролю в организации-исполнителе. При проведении нормоконтроля рекомендуется руководствоваться ГОСТ 2.111.

1.2. ГОСУДАРСТВЕННЫЕ СТАНДАРТЫ ОФОРМЛЕНИЯ НАУЧНЫХ РАБОТ

Требования по содержанию и оформлению научных работ регламентируются государственными стандартами, а именно:

ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления» [1].

ГОСТ 2.105-95 «Общие требования к текстовым документам. Единая система конструкторской документации. Межгосударственный стандарт» [2].

Эти два стандарта содержат наиболее полную информацию по оформлению работ. Многие положения в них и в тех, которые приведены ниже, повторяют друг друга.

ГОСТ 7.1-84 «Библиографическое описание документа. Общие требования и правила составления» [3].

ГОСТ 7.1-2003 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления» [4].

ГОСТ 7.82—2001 «Библиографическая запись. Библиографическое описание электронных ресурсов» [6].

Эти три ГОСТа определяют порядок оформления и вид библиографического описания, приведенного в работе.

ГОСТ 7.80-2000 «Библиографическая запись. Заголовок. Общие требования и правила составления» [5].

ГОСТ 7.0.11-2011 «Диссертация и автореферат диссертации. Структура и правила оформления» [7].

Стандарт 7.0.11-2011 появился сравнительно недавно – в 2011 году. Разработанный стандарт является полезным, хотя и содержит информацию из вышеприведенных ГОСТов.

1.3. СТРУКТУРА И ПРАВИЛА ОФОРМЛЕНИЯ ДОКУМЕНТОВ (НАУЧНЫХ РАБОТ)

Обобщая сведения и положения вышеприведенных стандартов, можно заметить следующее.

Содержание научных работ должно включать несколько разделов, которые приведены в таблице 1.1. Причем, некоторые разделы могут быть исключены из работы, например, «Приложения», а какие-то добавлены. Например, в реферат можно добавить раздел «Обозначения и сокращения».

1.3.1. Титульный лист

Для реферата. В верхней части титульного листа указывается организация, в которой выполняется работа. Далее буквами увеличенного кегля указывается тип («Реферат») и тема работы, ниже в правой половине листа — информация, кто выполнил и кто проверяет работу. В центре нижней части титульного листа пишут город и год выполнения реферата.

Таблица 1.1 - Структурные элементы научных работ

	Реферат	***ВКР и диссертация***	***НИР***
1	Титульный лист	Титульный лист	Титульный лист
2		Аннотация	Список исполнителей
3			Реферат
4	Оглавление	Оглавление	Содержание
5			Нормативные ссылки
6			Обозначения и сокращения
7	Введение	Введение	Введение
8	Основная часть (разделы, части)	Основная часть	Основная часть
9	Выводы (заключительная часть)	Заключение	Заключение
10	Список использованной литературы	Библиографический список	Список использованных источников
11	Приложения	Приложения	Приложения

Для диссертации.

- Полное наименование учебного заведения или научной организации.
- Фамилия, имя и отчество диссертанта (в именительном падеже).
- Заголовок работы (без слова «тема» и кавычек). Допускается использование уточняющего подзаголовка.
- Шифр из номенклатуры специальности работы и учёной степени, на соискание которой представляется диссертация.
- Ближе к правому краю— учёные звания, степень, фамилия и инициалы научного руководителя или консультанта.
- В нижнем поле— место и год выполнения диссертации (при этом год защиты тот, в котором диссертация сдана на предварительную экспертизу).

Для НИР.

- Наименование вышестоящей организации.
- Наименование организации-исполнителя НИР.
- Индекс Универсальной десятичной классификации (УДК).
- Коды Высших классификационных группировок Общероссийского классификатора промышленной и сельскохозяйственной

продукции для НИР (ВКГОКП), предшествующих постановке продукции на производство.

- Номера, идентифицирующие отчет.
- Грифы согласования и утверждения.
- Наименование работы.
- Наименование отчета.
- Вид отчета (заключительный, промежуточный).
- Номер (шифр) работы.
- Должности, ученые степени, ученые звания, фамилии и инициалы руководителей организации-исполнителя НИР, руководителей НИР.
- Место и дату составления отчета.

Для НИР приводят список исполнителей и реферат.

Список исполнителей

В список исполнителей должны быть включены фамилии и инициалы, должности, ученые степени, ученые звания руководителей НИР, ответственных исполнителей, исполнителей и соисполнителей, принимавших творческое участие в выполнении работы.

Реферат

Реферат должен содержать:

— Сведения об объеме отчета, количестве иллюстраций, таблиц, приложений, количестве частей отчета, количестве использованных источников.

— Перечень ключевых слов. Это от 5 до 15 слов или словосочетаний из текста отчета, которые в наибольшей мере характеризуют его содержание и обеспечивают возможность информационного поиска. Ключевые слова приводятся в именительном падеже и печатаются строчными буквами в строку через запятые.

— Текст реферата. Он должен отражать: объект исследования или разработки, цель работы, метод или методологию проведения работы, результаты работы, основные конструктивные, технологические и технико-эксплуатационные характеристики, степень внедрения; область применения, экономическую эффективность или значимость работы, прогнозные предположения о развитии объекта исследования.

1.3.2. Оглавление (содержание)

В оглавлении приводятся заголовки всех разделов работы (кроме подзаголовков, даваемых в подбор с текстом) и указываются номера страниц, с которых они начинаются. Заголовки должны полностью совпадать с присутствующими в основном тексте.

Заголовки должны располагаться в виде древовидной структуры (с соответствующими алфавитно-цифровыми индексами и отступами),

начинаться с прописной буквы, соединяться с номером страницы отточием (без дополнительной точки в конце).

1.3.3. Введение

Реферат

Во введении к реферату следует отразить:

- место рассматриваемого вопроса в естественнонаучной проблематике;
- теоретическое и прикладное значение рассматриваемого вопроса;
- обоснование выбора данной темы, коротко рассказать о том, почему именно она заинтересовала автора).

Диссертация, ВКР

Начало диссертации обычно повторяет её автореферат и предназначено для ознакомления с важными квалификационными характеристиками работы:

- актуальность, цели и задачи исследования;
- новизна и достоверность предложенных методов и решений;
- практическая и научная значимость, положения, выносимые на защиту;
- апробация работы и личный вклад соискателя;
- объём и структура диссертации.

НИР

Введение НИР должно содержать оценку современного состояния решаемой научно-технической проблемы, основание и исходные данные для разработки темы, обоснование необходимости проведения НИР, сведения о планируемом научно-техническом уровне разработки, о патентных исследованиях и выводы из них, сведения о метрологическом обеспечении НИР. Во введении должны быть показаны актуальность и новизна темы, связь данной работы с другими научно-исследовательскими работами.

Во введении промежуточного отчета по этапу НИР должны быть приведены цели и задачи этапа исследований, их место в выполнении НИР в целом. Во введении заключительного отчета о НИР помещают перечень наименований всех подготовленных промежуточных отчетов по этапам и их инвентарные номера.

1.3.4. Основная часть

Основная часть должна излагаться в соответствии с планом, четко и последовательно. В тексте должны быть ссылки на использованную литературу. При дословном воспроизведении материала каждая цитата должна иметь ссылку на соответствующую позицию в списке

использованной литературы с указанием номеров страниц, например /12, с.56/ или "В работе [11] рассмотрены…" .

В *реферате* достаточно двух глав.

Диссертация ВКР, НИР обычно содержит не менее трёх глав. В них подробно рассматривают методику и технику исследования. Содержание глав должно точно соответствовать теме работы. Структура должна примерно соответствовать указанным во введении задачам исследования.

Первая глава включает обзор предметной области, обоснование актуальности темы, указание задач, выбор направления исследований, включающий обоснование направления исследования, методы решения задач и их сравнительную оценку, описание выбранной общей методики.

Последующие главы должны отражать:

- процесс теоретических и (или) экспериментальных исследований, включая определение характера и содержания теоретических исследований, методы исследований, методы расчета, обоснование необходимости проведения экспериментальных работ, принципы действия разработанных объектов, их характеристики;
- обобщение и оценку результатов исследований, включающих оценку полноты решения поставленной задачи и предложения по дальнейшим направлениям работ, оценку достоверности полученных результатов и их сравнение с аналогичными результатами отечественных и зарубежных работ, обоснование необходимости проведения дополнительных исследований, отрицательные результаты, приводящие к необходимости прекращения дальнейших исследований.

В конце каждой главы могут быть сделаны выводы по полученным результатам.

Каждая глава текста должна начинаться с нового листа, независимо от того, где окончилась предыдущая. Полглавы и параграфы с новой страницы НЕ начинают.

1.3.5. Заключение (Выводы, заключительная часть)

должны содержать краткое обобщение рассмотренного материала, выделение наиболее достоверных и обоснованных положений и утверждений, а также наиболее проблемных, разработанных на уровне гипотез, важность рассмотренной проблемы с точки зрения практического приложения, мировоззрения, этики и т.п.

1.3.6. Список используемой литературы

Литературные источники следует располагать в следующем порядке:

- энциклопедии, справочники;
- книги по теме реферата (фамилии и инициалы автора, название книги без кавычек, место издания, название издательства, год издания, номер (номера) страницы);

- газетно-журнальные статьи (название статьи, название журнала, год издания, номер издания, номер страницы).

Каждый включённый в список литературный источник должен быть отражён в рукописи диссертации. Не рекомендуется включать в этот список энциклопедии, справочники, научно-популярную литературу, жёлтую прессу.

Примеры библиографических записей можно посмотреть в соответствующих ГОСТтах [1, 3, 4].

1.3.7. Приложения

Это необязательная часть. Сюда выносятся все материалы, не являющиеся критичными для понимания сути диссертационной работы. Это могут быть листинги программ, громоздкие таблицы, графики, копии подлинных документов, отдельные положения из инструкций и правил и т.п.

1.3.8. Формат

Научные работы должны быть выполнены на одной стороне листа белой бумаги формата А4 (210х297 мм). Интервал межстрочный - полуторный. Цвет шрифта - черный. Гарнитура шрифта основного текста — «Times New Roman» или аналогичная. Кегль (размер) от 12 до 14 пунктов. Размеры полей страницы (не менее): левое — 25 мм, верхнее и нижнее — 20 мм, правое -10 мм. Формат абзаца: полное выравнивание («по ширине»). Отступ красной строки одинаковый по всему тексту.

Страницы должны быть пронумерованы с учётом титульного листа, который не обозначается цифрой. В работах используются цитаты, статистические материалы. Эти данные оформляются в виде сносок (ссылок и примечаний).

Расстояние между названием главы (подраздела) и текстом должно быть равно 2,5 интервалам. Однако расстояние между подзаголовком и последующим текстом должно быть 2 интервала, а интервал между строками самого текста — 1,5. Размер шрифта для названия главы — 16 (полужирный), подзаголовка — 14 (полужирный), текста работы — 14. Точка в конце заголовка, располагаемого в середине листа, не ставится. Заголовки не подчёркиваются. Абзацы начинаются с новой строки и печатаются с отступом в 1,25 сантиметра. Оглавление (содержание) должно быть помещено в начале работы.

1.3.9. Заголовки

Заголовки разделов и подразделов следует печатать на отдельной строке с прописной буквы без точки в конце, не подчеркивая. Например, ВВЕДЕНИЕ, ЗАКЛЮЧЕНИЕ.

ВВЕДЕНИЕ, ЗАКЛЮЧЕНИЕ, СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ не нумеруются.

Выравнивание по центру или по левому краю. Отбивка: перед заголовком — 12 пунктов, после — 6 пунктов. Расстояние между названием главы и последующим текстом должно быть равно двум междустрочным интервалам. Такое же расстояние выдерживается между заголовками главы и параграфа. Расстояния между строками заголовка принимают таким же, как и в тексте. Подчеркивать заголовки и переносить слова в заголовке не допускается.

Если документ имеет подразделы, то нумерация пунктов должна быть в пределах подраздела и номер пункта должен состоять из номеров раздела, подраздела и пункта, разделенных точками, например:

3 Методы испытаний

3.1 Аппараты, материалы и реактивы

3.1.1

3.1.2 Нумерация пунктов первого подраздела третьего раздела документа

3.1.3

3.2 Подготовка к испытанию

3.2.1

3.2.2 Нумерация пунктов второго подраздела третьего раздела документа

*3.2.3 *

1.3.10. Нумерация страниц

Страницы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту (титульный лист и оглавление включают в общую нумерацию). На титульном листе номер не проставляют. Номер страницы проставляют без точки в центре нижней части листа.

1.3.11. Списки и перечисления

Внутри пунктов или подпунктов параграфов могут быть приведены перечисления.

Перед каждым перечислением следует ставить дефис или, при необходимости ссылки в тексте документа на одно из перечислений, строчную букву (за исключением ё, з, о, г, ь, и, ы, ъ), после которой ставится скобка.

Для дальнейшей детализации перечислений необходимо использовать арабские цифры, после которых ставится скобка, а запись производится с абзацного отступа, как показано в примере.

Пример

*а)*_____

б) _____

*1)*_____

*2)*_____

в) _____

1.3.12. Иллюстрации

Иллюстрации (чертежи, графики, схемы, компьютерные распечатки, диаграммы, фотоснимки) следует располагать в отчете непосредственно после текста, в котором они упоминаются впервые, или на следующей странице, выровняв по центру листа.

Иллюстрации могут быть в компьютерном исполнении, в том числе и цветные.

На все иллюстрации должны быть даны ссылки в отчете.

Фотоснимки размером меньше формата А4 должны быть наклеены на стандартные листы белой бумаги.

Иллюстрации, за исключением иллюстраций приложений, следует нумеровать сквозной нумерацией арабскими цифрами.

Слово «рисунок» и его наименование располагают посередине строки.

Допускается нумеровать иллюстрации в пределах раздела. В этом случае номер иллюстрации состоит из номера раздела и порядкового номера иллюстрации, разделенных точкой. Например, Рисунок 1.1.

Иллюстрации, при необходимости, могут иметь наименование и пояснительные данные (подрисуночный текст). Слово «Рисунок» и наименование помещают после пояснительных данных и располагают следующим образом: *Рисунок 1 — Детали прибора*.

Иллюстрации каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения. Например, Рисунок А.3.

При ссылках на иллюстрации следует писать «... в соответствии с рисунком 2» при сквозной нумерации и «... в соответствии с рисунком 1.2» при нумерации в пределах раздела.

1.3.13. Таблицы

Таблицы применяют для лучшей наглядности и удобства сравнения показателей. Название таблицы, при его наличии, должно отражать ее содержание, быть точным, кратким. *Название таблицы следует помещать над таблицей слева, без абзацного отступа в одну строку с ее номером через тире.*

Таблицу следует располагать в отчете непосредственно после текста, в котором она упоминается впервые, или на следующей странице.

На все таблицы должны быть ссылки в отчете. При ссылке следует писать слово «Таблица» с указанием ее номера.

Таблицу с большим количеством строк допускается переносить на другой лист (страницу). При переносе части таблицы на другой лист (страницу) слово «Таблица» и ее номер указывают один раз справа над первой частью таблицы, над другими частями пишут слово «Продолжение» и

указывают номер таблицы, например, «Продолжение таблицы 1». При переносе таблицы на другой лист (страницу) заголовок помещают только над ее первой частью.

Таблицу с большим количеством граф допускается делить на части и помещать одну часть под другой в пределах одной страницы. Если строки и графы таблицы выходят за формат страницы, то в первом случае в каждой части таблицы повторяется головка, во втором случае — боковик.

Если повторяющийся в разных строках графы таблицы текст состоит из одного слова, то его после первого написания допускается заменять кавычками; если из двух и более слов, то при первом повторении его заменяют словами «То же», а далее — кавычками. Ставить кавычки вместо повторяющихся цифр, марок, знаков, математических и химических символов не допускается. Если цифровые или иные данные в какой-либо строке таблицы не приводят, то в ней ставят прочерк.

Цифровой материал, как правило, оформляют в виде таблиц. Пример оформления таблицы приведен на рисунке рис.1.2.

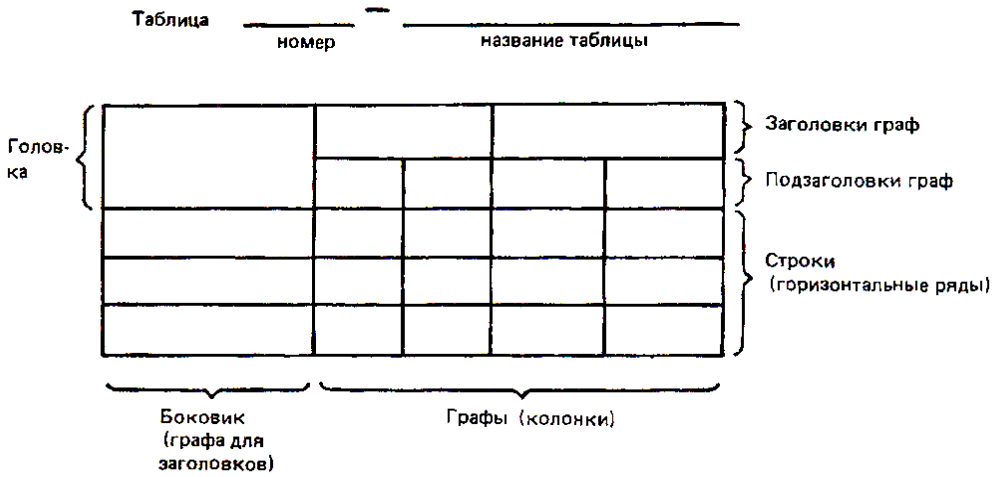


Рис. 1.2- Пример оформления таблицы

2. АЛГОРИТМИЗАЦИЯ

2.1 ОСНОВНЫЕ ЭТАПЫ РЕШЕНИЯ ЗАДАЧ НА ЭЛЕКТРОННО-ВЫЧИСЛИТЕЛЬНЫХ МАШИНАХ

Рассмотрим некоторые из определений раскрывающих суть процессов технологии программирования.

Под *технологией программирования* понимают методы разработки программ в совокупности с аппаратным и программным обеспечением для реализации задачи на персональном компьютере (ПК).

Программа – это набор операторов (команд), который может быть представлен как единое целое в некоторой вычислительной системе и который используется для управления поведением этой системы.

В дальнейшем под *вычислительной системой* мы будем понимать персональный компьютер. При этом следует различать понятия «Программа» и «Программирование». Суть понятия программирования не заключается только в написании программы.

В широком смысле, *программирование* – это все технические операции, необходимые для создания программы, включая анализ требований и все стадии ее разработки и реализации.

Алгоритм – это заранее заданная последовательность четко определенных правил или команд для получения решения задачи за конечное число шагов.

Любой алгоритм можно представить или записать тремя способами:

- *словесным (вербальным)* - с использованием слов и предложений,
- *табличным (аналитическим)* - с помощью формул и таблиц,
- *графическим* - с помощью рисунков, геометрических фигур и символов.

Самым наглядным из них является графический способ – представление алгоритма схемой.

Схема алгоритма – это графическое изображение его структуры, отдельных составных частей и взаимосвязей между ними.

Схема алгоритма представляет собой совокупность специальных символов (*блоков*), соединенных между собой стрелками, которые указывают последовательность его выполнения. Внутри каждого блока может быть записан его номер и краткое содержание исполняемых им функций (операций).

Решение задачи на ПК сводится к выполнению программы, введенной в память ЭВМ. Составлению программы предшествует разработка алгоритма решения задачи. Алгоритм реализует метод решения. Выбору метода решения предшествует анализ и формализация целевой задачи. Таким

образом, процесс подготовки задачи к решению на ПК можно представить в виде обобщенной схемы, представленной на рисунке рис.2.1.

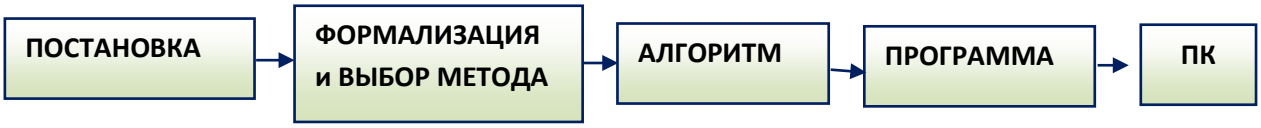


Рис.2.1 - Этапы решения задачи на ПК

Кратко рассмотрим содержание каждого представленного на ней этапа.

Постановка задачи

Постановка задачи содержит описание целей и условий ее решения. Иногда впервые поставленная задача требует разработки новых концепций и теорий, но гораздо чаще ситуация описывается в уже сложившихся терминах и понятиях.

Формализация задачи и выбор метода решения

Для обеспечения возможности решения задачи на ЭВМ она должна быть выражена так, чтобы было ясно, как ее реализовать на ПК, и каким методом реализации при этом воспользоваться. Для формализации задачи указываются исходные данные, начальные условия и необходимая точность вычисления, а также некоторые ограничения (например, на время решения).

Часто понятие формализации связывают с последовательным выражением условий задачи в математических терминах (формулах).

Выбор того или иного метода зависит от типа решаемой задачи.

Часто задача бывает такова, что ее точное решение получить невозможно. Можно получить только какое-либо приближенное решение с допустимой погрешностью. Для решения такого класса задач существуют разнообразные численные методы, которые рассматриваются в вычислительной математике. Они позволяют свести решение любой задачи к последовательному выполнению четырех арифметических действий и операций. Например, для решения систем обыкновенных дифференциальных уравнений первого порядка существует несколько численных методов решения (Адамса, Хэмминга, Рунге-Кутта и др.), которые отличаются временем решения и точностью. Из численных методов выбирается тот, который обеспечивает решение предъявленных к задаче требований при заданных ограничениях на ресурсы.

Разработка алгоритма программы

Формализация задачи позволяет выводить точные логические следствия из известных нам данных об исходном и конечном состоянии. Сравнительно редко путь, который ведет от исходного состояния к конечному состоянию, бывает сразу ясен. Как правило, он намечается в результате логического анализа формальной постановки задачи, выявленных новых свойств и отношений между понятиями и объектами, о которых в ней

идет речь. Сама задача разбивается на элементарные шаги и действия, выполняемые в дальнейшем на ПК. Этот путь решения задачи – последовательность действий, которые необходимо выполнить, чтобы от исходного состояния перейти к желаемому новому состоянию – называется алгоритмом решения задачи.

Алгоритм решения задачи может быть записан и предъявлен различными способами и с различной степенью детализации. Для представления алгоритма в виде, понятном для ПК, служат языки *программирования* (ЯП).

Написание программы и подготовка ее к вводу в ПК

Целью данного этапа является запись алгоритма на языке программирования и перенос текста программы на носитель, с которого она может быть введена в ПК.

Язык программирования (ЯП) – это система обозначений, служащая для точного описания программ или алгоритмов для ПК.

Языки программирования являются искусственными языками, в которых синтаксис и семантика строго определены. Поэтому при применении их по назначению они не допускают свободного толкования выражения, характерного для естественного языка.

Если ЯП выбран, то запись алгоритма на нем и перенос текста программы на носитель можно совместить, работая в *интегрированной среде программирования* данного ЯП. При выборе ЯП необходимо руководствоваться не только критерием простоты написания программы и сокращением сроков ее отладки. В тех случаях, когда создаются программы, предназначенные для многократного их использования, более важно обеспечить их высокую эффективность.

Отладка программы и ее выполнение на ПК

Тестирование программы - процесс поиска ошибок в программе.

Отладка программы - процесс устранения ошибок.

Основная цель данного этапа – выявление и исправление ошибок в программе и получение конечного результата решения задачи.

По существующим оценкам на отладку программист затрачивает до 40% времени, отводимого на разработку программы. Кратко, этап решения задачи на ПК состоит из трех шагов: трансляции, редактирования и выполнения. Эти шаги могут повторяться неоднократно при отладке программы до тех пор, пока получаемые результаты не будут соответствовать предъявляемым к задаче требованиям.

2.2. СПОСОБЫ ЗАПИСИ АЛГОРИТМА

Алгоритм характеризуют его *свойства* и *способ описания*.

2.2.1. Свойства алгоритма

Дискретность. Процесс получения конечного результата разбивается на отдельные этапы (дискретные шаги) так, что значения величин на каждом следующем шаге определяются значениями величин, полученных на шаге предыдущем.

Определенность. Каждое правило алгоритма должно быть четким и однозначным в его толковании и исполнении.

Результативность. Требуемый результат должен быть получен за конечное число шагов алгоритма решения задачи.

Массовость. Алгоритм в общем виде разрабатывается так, чтобы его можно было применить для класса задач, отличающихся друг от друга только исходными данными.

2.2.2. Способ описания алгоритма

Требования по оформлению схем алгоритмов, программ, данных и систем состоят из символов, имеющих заданное значение, краткого пояснительного текста и соединяющих линий. Эти требования регламентируются государственными стандартами:

ГОСТ 19.701-90 (ИСО 5807-85). Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. Единая система программной документации [8].

ГОСТ 19003-80. Схемы алгоритмов и программ. Обозначения условные графические. Единая система программной документации [9].

Схемы алгоритмов программ отображают последовательность операций в программе. Схемы могут использоваться на различных уровнях детализации, причем число уровней зависит от размеров и сложности задачи обработки данных. Уровень детализации должен быть таким, чтобы различные части и взаимосвязь между ними были понятны в целом.

2.2.3. Символы алгоритмов

Схема программы состоит из:

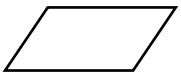
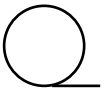
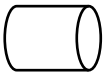
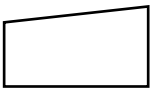
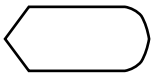
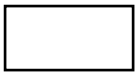
- символов процесса, указывающих фактические операции обработки данных (включая символы, определяющие путь, которого следует придерживаться с учетом логических условий);
- линейных символов, указывающих поток управления;
- специальных символов, используемых для облегчения написания и чтения схемы.

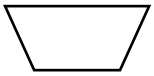
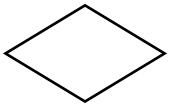
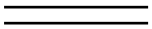
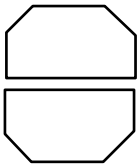
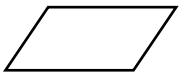
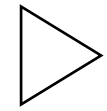
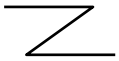
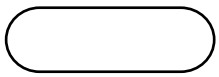
Основные символы представлены в таблице 2.1, ниже.

Все символы схемы алгоритма или технологического процесса обработки информации должны иметь лаконичные и ясные пояснения:

- в символах операций проставляются их названия;
- в символах машинных носителей — сокращенные наименования и идентификаторы соответствующих массивов или файлов;
- в символах информации, выводимой на печать или экран, - наименования ведомостей, машинограмм или их идентификаторы.

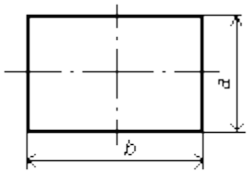
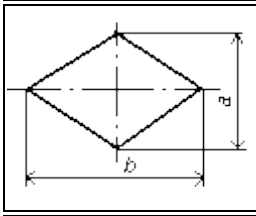
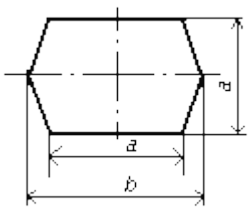
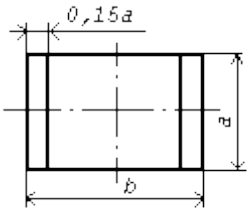
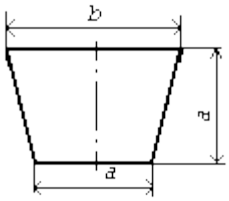
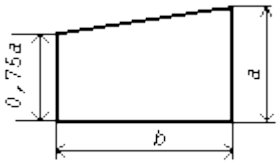
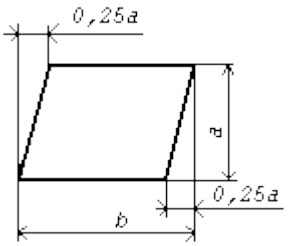
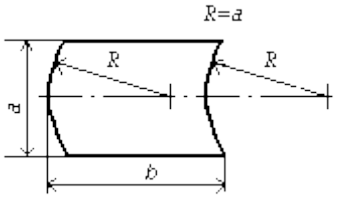
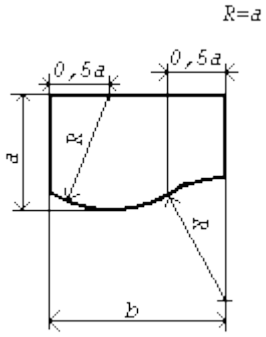
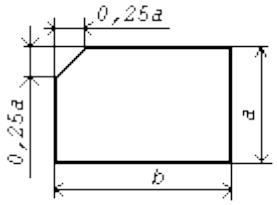
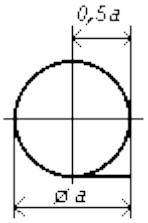
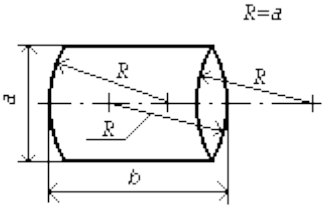
Таблица 2.1 – Символы изображения алгоритмов

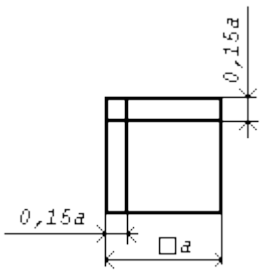
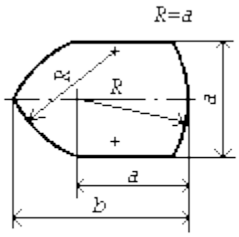
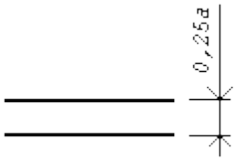
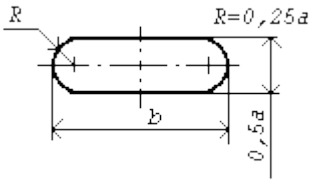
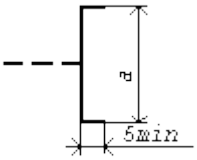
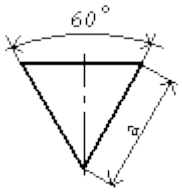
Графическое изображение	**Название символа**	**Описание символа**
Символы данных		
	Данные	Данные, носитель данных не определен
	Запоминаемые данные	Хранимые данные в виде, пригодном для обработки, носитель данных не определен
	Оперативное запоминающее устройство	Данные, хранящиеся в оперативном запоминающем устройстве
	Запоминающее устройство с последовательным доступом	Данные, хранящиеся в запоминающем устройстве с последовательным доступом (кассетная магнитная лента и т. д.)
	Запоминающее устройство с прямым доступом	Данные, хранящиеся в запоминающем устройстве с прямым доступом (жесткий магнитный диск, гибкий магнитный диск, оптический диск и т. д.)
	Документ	Данные, представленные на носителе в удобочитаемой форме (машинограмма, документ для оптического или магнитного считывания, микрофильм и т. д.)
	Ручной ввод информации	Данные, вводимые вручную с устройств любого типа (клавиатура, кнопки, световое перо и т. д.)
	Дисплей	Данные, представленные на экране для визуального отображения
Символы процесса		
	Процесс	Функция обработки данных любого вида

	Предопределенный процесс	Функция, состоящая из одной или нескольких операций или шагов программы, которые определены в другом месте
	Ручная операция	Процесс, выполняемый человеком
	Решение	Процесс переключательного типа, позволяющий выбрать один из нескольких альтернативных выходов
	Параллельные действия	Синхронизация двух или более параллельных операций
	Границы цикла	Циклический процесс, символы отображают начало и конец цикла
Символы линий		
	Линия	Отображает поток данных или управления. При необходимости могут быть добавлены стрелки-указатели
	Передача управления	Отображает непосредственную передачу управления от одного процесса к другому
	Канал связи	Отображает передачу данных по каналу связи
	Терминатор	Отображает начало или конец схемы
	Пропуск	Используют для отображения пропуска символа или группы символов. Он применяется главным образом в схемах, изображающих общие решения с неизвестным числом повторений

Кроме того, следует соблюдать размеры указанных символов. Для некоторых из них информация приведена в таблице 2.2.

Таблица 2.2. – Размеры символов алгоритмов

Наименование	**Обозначение и размеры в мм**	**Наименование**	**Обозначение и размеры в мм**
Процесс *		Решение *	
Модификация *		Предопределенный процесс *	
Ручная операция *		Ручной ввод *	
Данные *		Запоминаемые данные (неавтономная память) *	
Документ		Перфокарта	
Запоминающее устройство с последовательным доступом (магнитная лента)		Запоминающее устройство с прямым доступом (магнитный барабан)	

Оперативная память		Дисплей	
Параллельные действия		Терминатор *	
Комментарий		Передача управления	

Размер *a* должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер *a* на число, кратное 5. Размер *b* равен *1,5a.*

При ручном выполнении схем алгоритмов и программ для символов, обозначенных в таблице звездочкой ***,** допускается устанавливать *b* равным *2a*.

Заметим, что схемы символов можно взять из Ms.Word, Ms.Excel, используя вставку «Иллюстрации/Фигуры», пакет Visio и т.д.

2.3.БАЗОВЫЕ (ТИПОВЫЕ) АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ

Каждому элементарному ВП соответствует своя *базовая (типовая) алгоритмическая конструкция (БАК).* Логика алгоритма и программы должна опираться на минимальное число достаточно простых БАК.

Любой реальный вычислительный процесс произвольной сложности может быть представлен в виде некоторой комбинации базовых алгоритмических конструкций.

К базовым алгоритмическим конструкциям относятся:

- *линейная последовательность;*
- *ветвление (выбор);*
- *цикл (повторение).*

Последние две базовые конструкции могут использоваться в различных модификациях.

Линейная последовательность – предполагает последовательное выполнение операторов: сначала первый, потом второй и т.д.

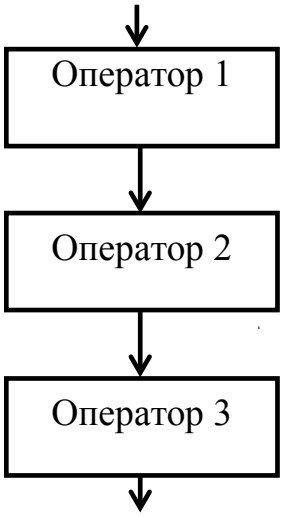


Рис.2.2 - Линейный алгоритм

Ветвление – алгоритмическая конструкция, в которой порядок выполнения операторов определяется исходя из некоторого заданного условия. Если условие выполняется, то для выполнения программы выбирают один путь, если нет – то другой.

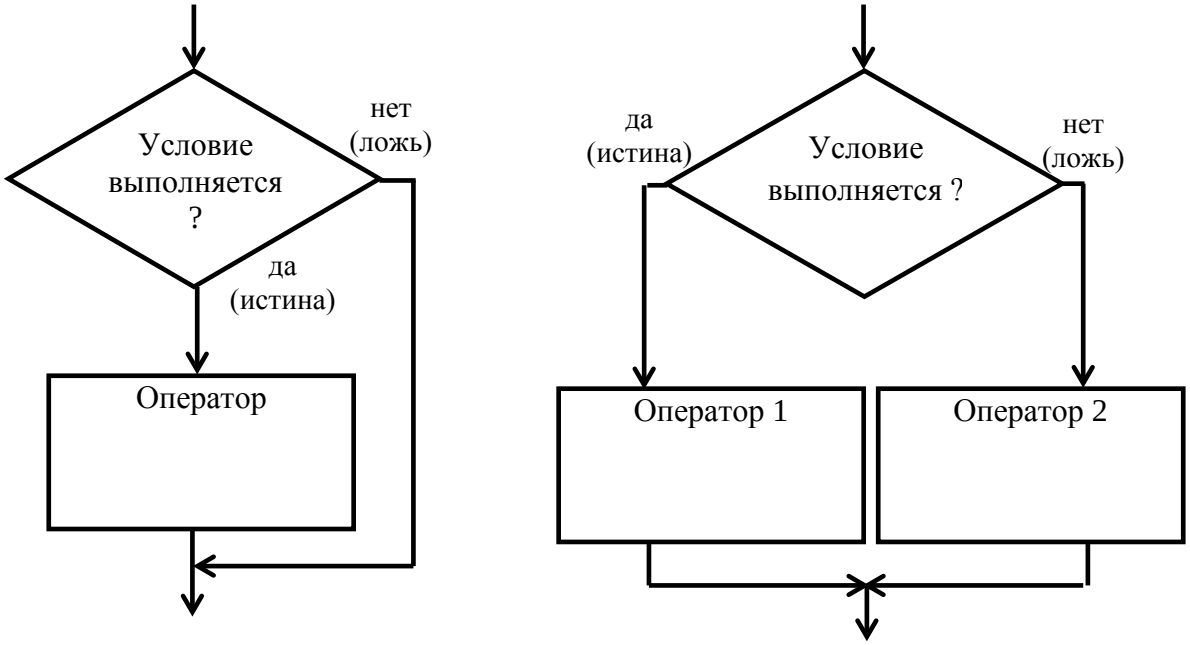


Рис.2.3 – Ветвления

Цикл – алгоритмическая конструкция, которая обеспечивает *повторное выполнение некоторой группы операторов* заданное число раз. Эта группа операторов называется *телом цикла.* Количество обращений к телу цикла

определяется значением *параметра цикла*. Начальное значение этого параметра задают перед обращением к циклу. При каждом выполнении *тела цикла параметр цикла* изменяют и при новом выполнении цикла проверяют условие соответствия значения *параметра цикла* некоторой постоянной величине. Последняя может иметь различный тип: числовой, текстовый, логический. Главное, чтобы в результате выполнения проверки условия было выработано решение – условие выполняется (истина) или не выполняется (ложь).

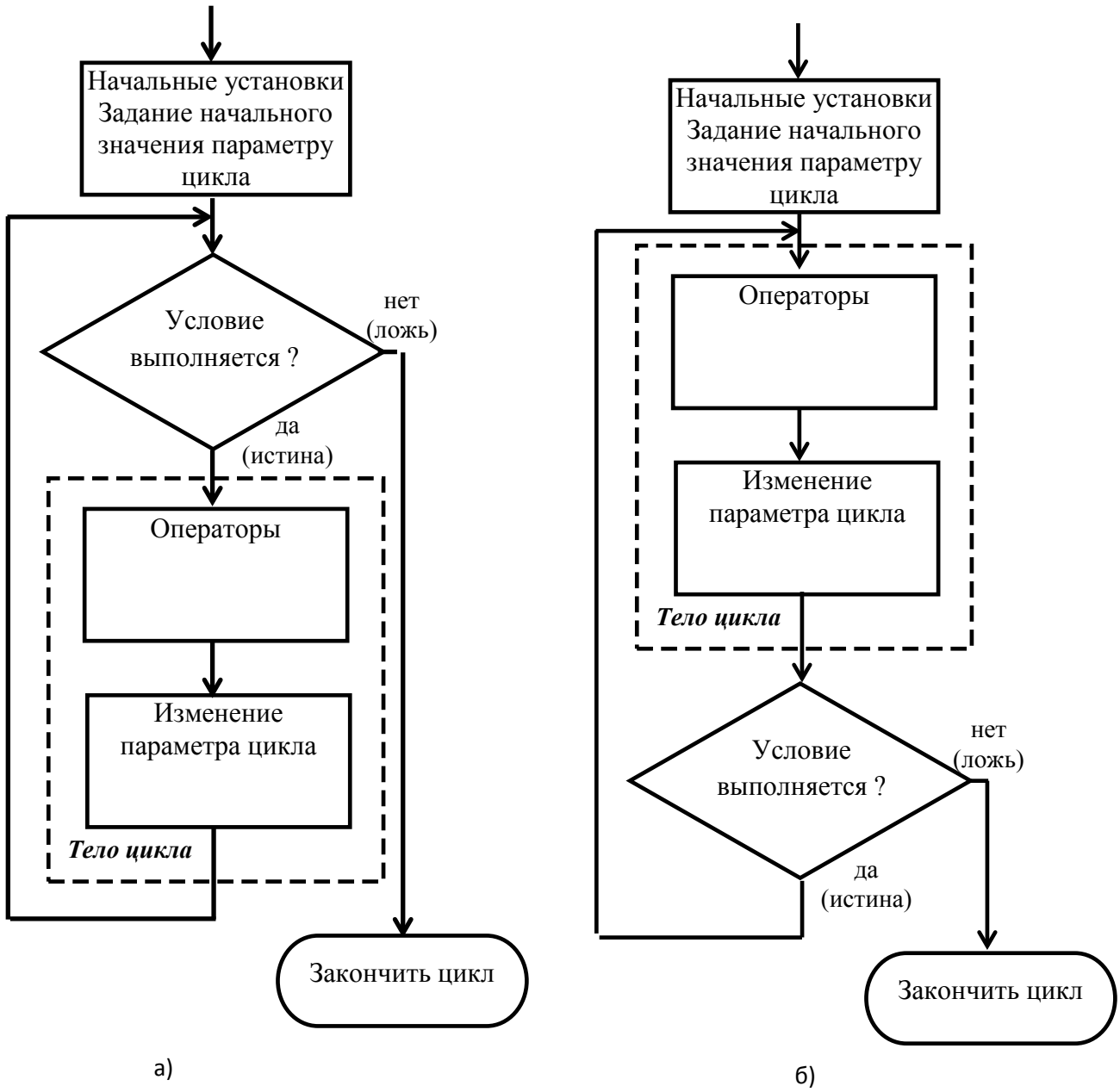


Рис. 2.4 - Структура циклов с предусловием (а) и постусловием (б)

По структуре различают следующие типы циклов:

1. *с предусловием* (проверка окончания выполнения цикла предшествует телу цикла);
2. *с постусловием* (тело цикла предшествует проверке условия окончания цикла).
3. *параметрический цикл*

Схематично эти структуры соответственно выглядят так, как показано на рисунках рис.2.4. и рис 2.5.

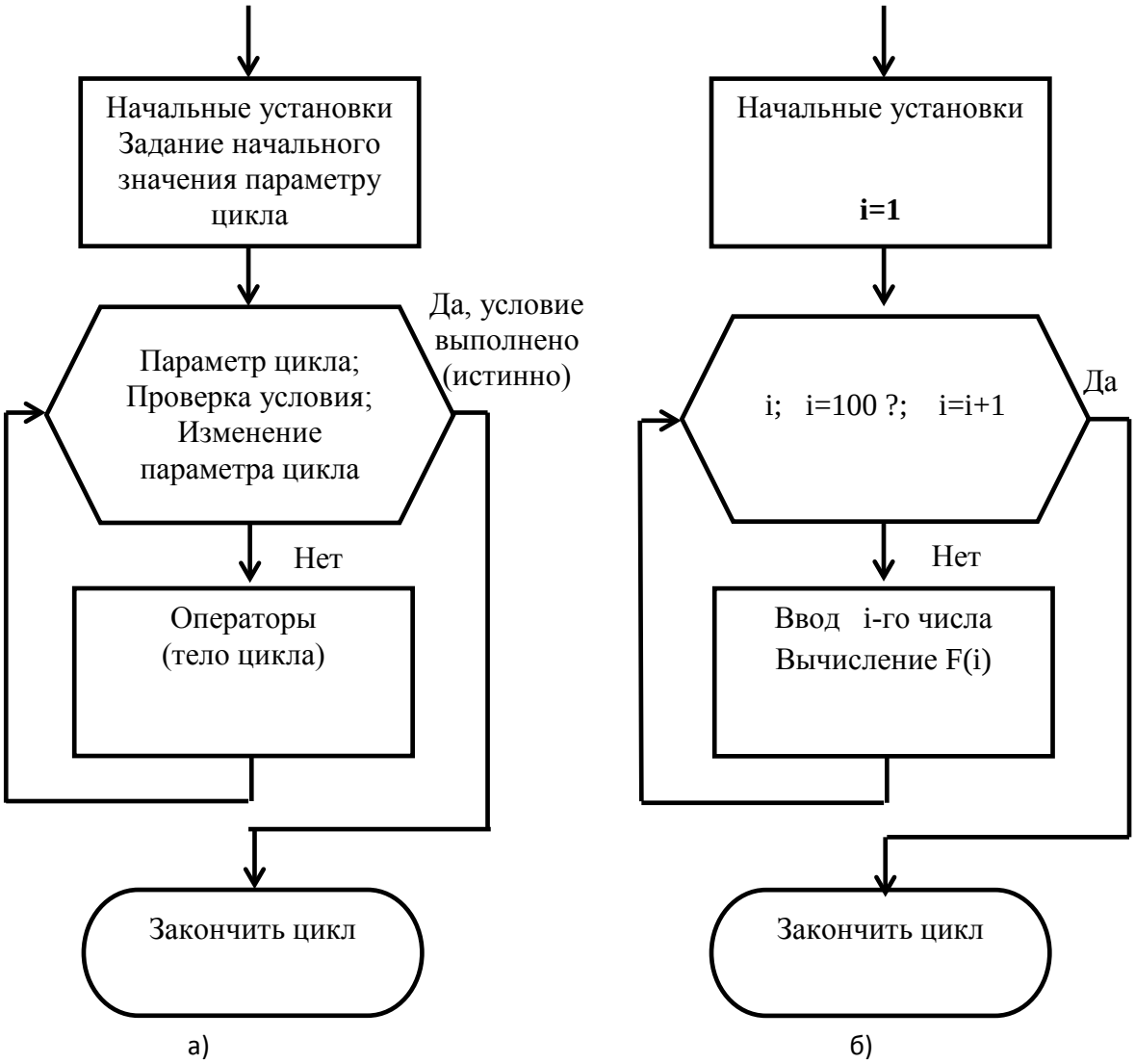


Рис. 2.5 - Структура параметрического цикла. (а) – в общем виде, (б) - с примером

2.4. ПРИМЕР ПОЭТАПНОГО РЕШЕНИЯ КОНКРТНОЙ ЗАДАЧИ НА ЭВМ

Как было показано в подразделе 2.1, решение задачи на ПК можно разбить на 5 этапов. Этапы 1-2-3 выполняются на бумаге при подготовке

задачи к решению на ПК, и от степени их проработки зависит качество конечного решения. Конкретный элементарный пример, приведенный ниже, позволяет представить содержание этих этапов.

Пример

Этап 1. Постановка задачи.

Определить и вывести на печать дискриминант решения квадратного уравнения вида: $y=ax^2+bx+5$, если его значение положительно. Значения величин *a* и *b* задаются с клавиатуры. Если величина дискриминанта отрицательна, то на экран выводится сообщение об этом.

Этап 2. Формализация и выбор метода.

1) Начальными значениями являются величины сторон *a, b.* В постановке задачи не сказано, какого типа эти числа – целые или вещественные. Так как целые числа являются подмножеством вещественных чисел, то целесообразно приписать значениям *a, b* вещественный тип.

2) Известно из математики, что для нахождения дискриминанта используется формула $D = b^2 - 4ac$.

3) Из математического метода решения следует, что для программной реализации необходимо определить еще одну величину: для идентификации дискриминанта – величину *D.* Так как при ее вычислении основу составляют числа *a, b*, которые имеют вещественный тип, то и тип числа *D* также должен быть вещественным.

4) Так как в процессе вычислений потребуется осуществлять проверку знака дискриминанта, и, в зависимости от него, выводить либо значение числа *D,* либо сообщение о его отрицательности, то в структуре алгоритма следует использовать базовую алгоритмическую конструкцию типа «ветвление», которая в программе может быть реализована с помощью условного оператора.

Этап 3. Структурная блок-схема алгоритма.

Из формализации задачи следует следующая последовательность действий:

Шаг 1. Определяются величины, участвующие в вычислениях и их тип.

Шаг 2. Водятся числа *a, b.*

Шаг 3. По формуле: $D = b^2 - 4ac$ вычисляется дискриминанта.

Шаг 4. Задается условие проверки.

Шаг 5a. Если условие выполняется, то выводится значение *D.*

Шаг 5b. Если условие не выполняется, то выводится сообщение об отрицательности дискриминанта.

Блок-схема реализации данного алгоритма с использованием базовых управляющих структур и соответствующих символов представлена на рис.2.6.

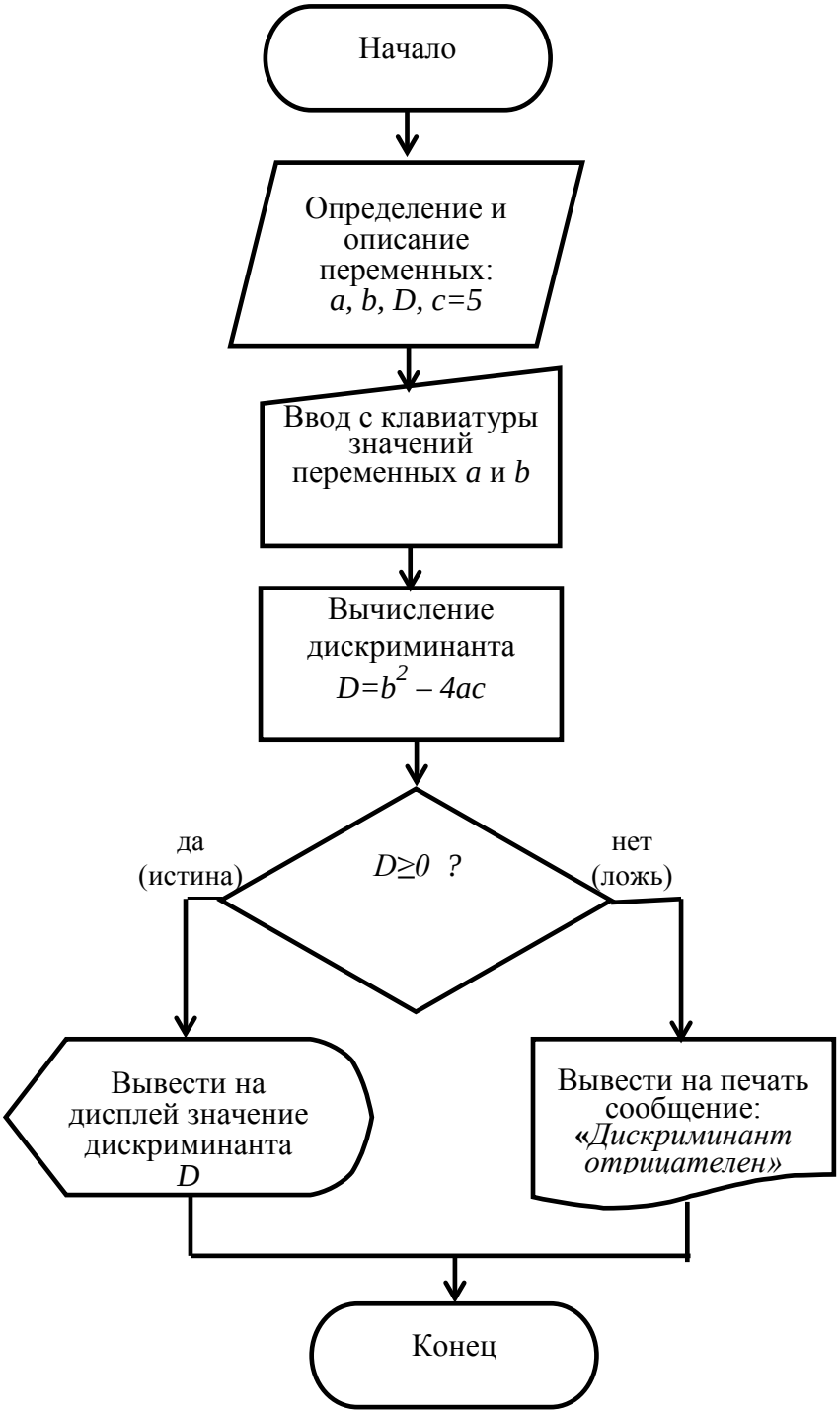


Рис.2.6- Алгоритм расчета дискриминанта и выхода данных

Задание 2.1

Произвести формализацию решения задач и составить блок-схему алгоритма их решения:

1. Дан треугольник со сторонами, величины которых соответственно равны числам *a, b, c***.** Определить высоты этих сторон и площадь треугольника (использовать формулу Герона). Значения сторон

ввести в ПК с клавиатуры. Вывод их высот после вычислений осуществить на экран монитора.

2. Ввести с клавиатуры массив из трех целых чисел и подсчитать сумму элементов этого массива после окончания ввода. Если значение суммы меньше числа 100, то вывести на экран сообщение об этом. Если значение суммы больше или равно числу 100, то вывести полученное значение на принтер.

3. Известно, что ежемесячный платеж за платную стоянку автомобиля составляет *X* рублей. С другой стороны, стоимость гаража, приобретенного в личное пользование, составит *Y* рублей, и, кроме того, придется платить ежегодный взнос *Z* рублей. Докажите, что покупка гаража выгоднее, чем использование платной стоянки. Для этого определите через сколько лет (до десятых) наступит момент, когда суммарные стоимости за гараж и стоянку, определенные нарастающим итогом, сравняются.

3. ПРОГРАММИРОВАНИЕ

Язык *C++* - *это универсальный язык программирования*, способный реализовать объектно-ориентированный подход. Двумя его основными предшественниками были языки *C* и *Симула*. Важнейшие свойства объектно-ориентированного программирования в языке *C++* являются развитием идей, заложенных в языке *Симула.*

Язык *C* был создан *Деннисом Ричи* прежде всего для операционной системы *Unix*. Постепенно *C* приобрел широкое признание, появились его реализации для практически всех операционных систем, и он стал стандартным языком системного и прикладного программирования.

От C язык C++ унаследовал эффективность, необходимую для системного программирования, начиная от написания драйверов внешних устройств до разработки компиляторов и систем управления базами данных.

Совместимость с *C* обусловила еще одну важную область применения *C++* - программирование сложных графических пользовательских интерфейсов. Интерфейсы прикладных программ с графической средой Microsoft Windows были разработаны, прежде всего, для *C*. С развитием и распространением *C++* разработка графических пользовательских интерфейсов перешла к нему.

К настоящему времени ситуация с использованием различных языков продолжает меняться. Для разработки простых интерфейсов и решения прикладных задач стали использоваться такие программные средства как Visual Basic, PowerBuilder. В последнее время все более широко стал употребляться язык Java. Тем не менее, *C++* позволяет разработать наиболее сложные программы, поскольку с его помощью можно использовать всю мощь базовой графической среды.

3.1. СОСТАВ СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Система программирования *(СП)* представляет собой комплекс программ, обеспечивающих автоматизацию программирования, отладку программ и их сопровождение. Большое количество программ, входящих в систему, позволяет называть ее интегрированной системой программирования *(ИСП*). *СП* включает в себя:

язык программирования,
библиотеку стандартных программ,
текстовый редактор,
компилятор,
компоновщик,
отладчик.

Языком программирования (ЯП) называют определенный набор символов и правил, устанавливающих способы комбинации этих символов для записи программ.

Редактор текста – это программа, реализующая функции создания и редактирования текстовых файлов.

Библиотека стандартных программ включает часто используемые в разных программах подпрограммы, оформленные по единым правилам.

Компилятором (транслятором) называется системная программа, обеспечивающая перевод текста программы на язык высокого уровня в машинный код. Компиляция сопровождается синтаксическим контролем и выдачей сообщений об ошибках. В результате получается *объектный модуль*. Откомпилированный текст программы, если даже все ошибки устранены, не готов к выполнению. Необходимо установить внутренние связи в программе, связать внешние ссылки с библиотекой стандартных программ. Результирующая программа может состоять из нескольких модулей, которые необходимо объединить в один. Все эти функции выполняет программа, называемая *компоновщиком (редактором связей).*

Загрузчик записывает полученный модуль в ОП и передает управление на его выполнение.

Отладчик – программа, служащая для отладки исходной программы на уровне входного языка. Отладчик контролирует ход выполнения программы, приостанавливает ее в контрольных точках, а также организует пошаговый режим (трассировку), позволяя прерывать работу программы после выполнения каждой команды.

Работу такой системы по созданию программы можно представить схемой, изображенной на рисунке рис.3.1:

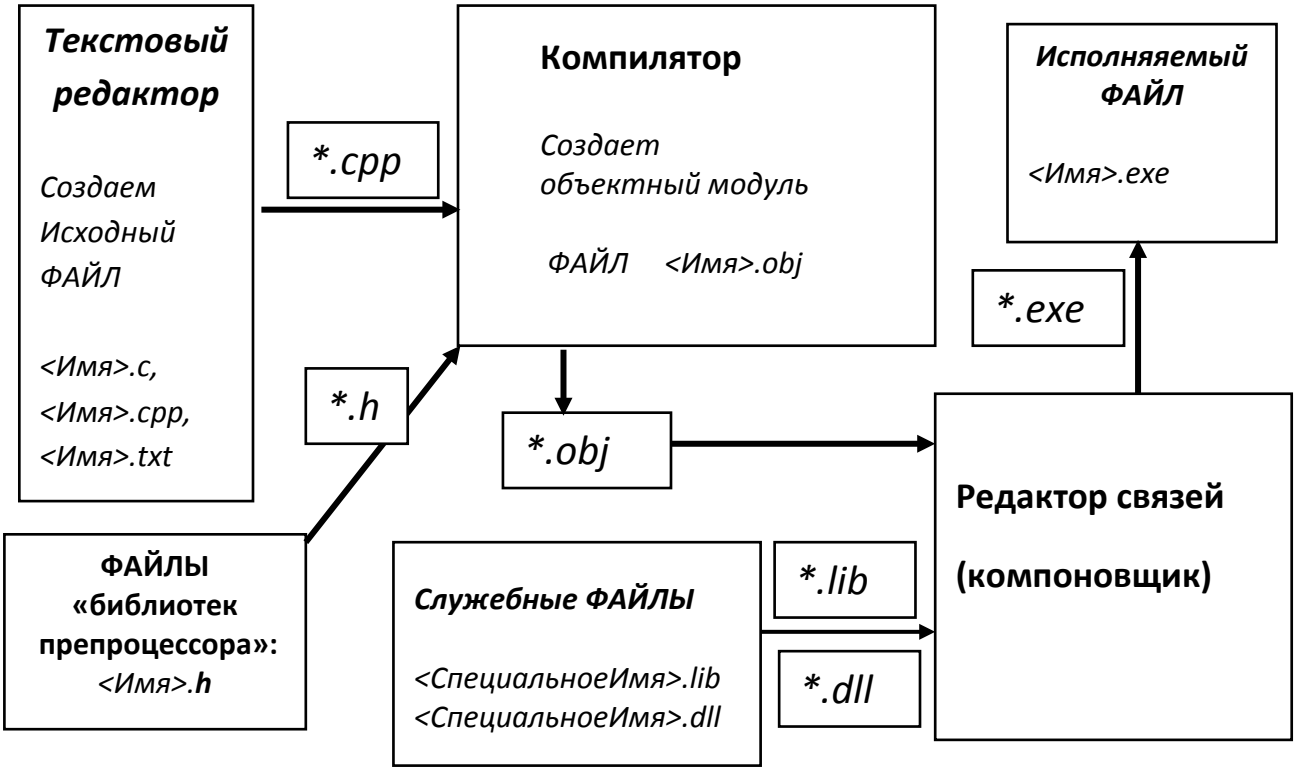


Рис.3.1 - Работа системы программирования

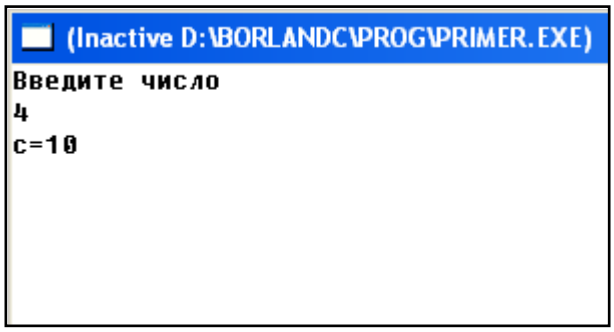
Из сказанного выше следует различать такие понятия, как «исходный текст программы на алгоритмическом ЯП», «объектный код программы» и «программа – исполняемый код (файл)».

3.2 СТРУКТУРА ПРОГРАММЫ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ С

Одним из способов изучения ЯП является изучение языка на примерах составления простейших программ с подробным разбором их содержания. Практически, во всех учебниках по программированию на *C* [11,12,13] в качестве такой первой программы рассматривают примерно такую, как приведена ниже на рис 3.2.

```
// progr1.cpp
#include<iostream.h>                    //(1)
void main()                             //(2)
{ int a, b, c;                          //(3)
   cout<<"Введите число\n";             //(4)
   cin>>a;                              //(5)
   b = 6;                               //(6)
   c = a+b;                             //(7)
   cout<<"c="<<c;                       //(8)
}
```

(а)



(б)

Рис.3.2 – Листинг программы (а) и результат ее выполнения (б)

Отметим! В настоящее время на рынке программных продуктов компиляторы с языка C++ представлены достаточно широко. При этом следует учитывать их некоторые особенности, а также действие стандарта. В 1998 г. действовал стандарт ISO/IEC 14882, сейчас действует новый стандарт, принятый в 2003г [14].

Листинг приведенной программы предполагает использование компилятора Borland C++ 3.1, который используется и в классах факультета технологического менеджмента и инноваций.

Ниже будет приведен вариант этой же программы для случая использования среды программирования Microsoft Visual C++ 2005 (2008, 2010). Там же будут отмечены отличия исходного текста программы для этих вариантов.

Итак, рассмотрим эту программу. После двойного слеша следуют комментарии, которые не обрабатываются компилятором. Цифрами в скобках проставлены номера строк.

(1) В первой строке программы #include <iostream.h> записана директива препроцессора.

Препроцессор – это составная часть пакета *ИСП С*, которая обрабатывает исходный текст программы до того, как он пройдет через компилятор. Препроцессор работает на первом шаге компиляции программ на *С* и подключает указанные файлы.

В нашем примере мы подключили файл библиотеки *iostream.h*. В отличии от обычной библиотеки, в библиотеках систем программирования содержатся стандартные функции и средства, в данном случае – стандартные средства экранного вывода С++ - *cout* и *cin.* На этапе компиляции содержимое включенного файла просто вставляется в исходный текст программы. Если бы мы не подключили файл *iostream.h,* то пользоваться этими средствами ввода/вывода было бы нельзя.

Итак, команды препроцессора программе начинаются с символа «#». Различают две команды: команду включения файла – *#include* и команду замещения (замены) лексических единиц – *#define.*

Запомним, что любая строка вида *#include< имя_файла >* или *#include* "*имя_файла"* заменяется содержимым файла с именем *«имя_файла*». Если имя соответствующего файла заключено в кавычки, то файл ищется среди исходных файлов пользовательской программы. Если такового не оказалось или имя файла заключено в угловые скобки (< >), то поиск осуществляется в системных библиотеках.

Признаком системной библиотеки является ее расширение, состоящее из одной буквы *h* после точки. Например, имя *math.h* означает системную библиотеку математических функций. Такой способ позволяет собрать вместе модули большой программы, размещенные в разных физических файлах. Он гарантирует, что все исходные файлы будут пользоваться одними и теми же определениями, и описаниями переменных, благодаря чему уменьшается количество ошибок.

Команда замещения лексических единиц имеет вид:

#define идентификатор подстановка

Она вызывает замену в оставшейся части программы названного идентификатора на текст подстановки. Например, подстановка вида *#define N 1000* заменяет каждое вхождение идентификатора *N* в тексте программы на число *1000.*

Для определения и описания препроцессорных директив в программах существуют ограничения. Во-первых, препроцессорная директива размещается в одной строке. Во-вторых, символ «#», вводящий каждую директиву препроцессора, должен быть первым отличным от пробела символом в строке с препроцессорной директивой. В-третьих, как правило, препроцессорные директивы объявляются в начале программы.

(2) Во второй строке программы void main() записана главная функция, содержащая собственно тело программы. В программе на языке *C/C++* может быть множество различных функций или, в простейшем случае, они могут отсутствовать. Но главная функция *main()* должна быть всегда.

(3) Далее описаны переменные *a, b, c,* которые, судя по их типу *(int)* могут принимать только целочисленные значения.

Если мы заходим, чтобы эти переменные могли принимать и вещественные значения, то необходимо записать:

float a, b, c;

В любом случае, в конце объявления последней переменной необходимо поставить точку с запятой. Имена этих переменных называют *идентификаторами.*

(4) В следующей строке, с помощью инструкции *cout<<"Введите число\n";* выводим на экран текст «*Введите число».*

Эта инструкция осуществляет вывод строки символов, заключенной в кавычки, в *выходной поток* cout, который автоматически связывается с экраном монитора, когда программа начинает выполняться.

Выходной поток – это отдельный процесс, обеспечивающий вывод данных на указанный терминал (по умолчанию на экран). Двойные угловые открывающиеся скобки << - это оператор вывода данных в поток.

Сочетание *\n* – это управляющая константа, которая означает перевод строки (курсора) в начало следующей строки (вертикальная табуляция). Этот управляющий символ всегда берется в кавычки.

!! Каждая инструкция заканчивается точкой занятой.

(5) В пятой строке мы присваиваем начальные значения переменным *a* и *b,* причем значение переменной *a* вводим с клавиатуры, используя оператор *cin>>a;*.

Пока пользователь не введет число, на экране будет виден мигающий знак курсора. Оператор ввода >>, который связан со стандартным потоком ввода *cin* (обеспечивает ввод данных с клавиатуры в указанное место программы). Получив данную инструкцию, программа считывает с клавиатуры символ, который будет присвоен переменой *a.* В нашем случае переменная *a* – целочисленная (*mun int),* поэтому необходимо вводить целое число.

Далее *(в строках 6 и 7*) происходит сложение величин двух переменных, результат записывается в переменную *c.*

(8) Последняя, не считая скобки, строка cout<<"c="<<c выводит на экран сообщение *«c=10».* Из этой инструкции следует, что:

- оператор вывода << , как и всякий оператор, может повторяться в выражении языка С ++ сколько угодно раз;
- выражение после оператора вывода << , заключенное в кавычки , воспринимается как текст *(<<"c=")*, без кавычек – позволяет вывести значение переменной *(<<c).*

3.3.ИНТЕГРИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ ДЛЯ КОМПИЛЯТОРА BORLAND C++ 3.1

3.3.1 Запуск приложений

Запуск консольного приложения производится командой

C:\DORLAND\BIN\BC.EXE

Запуск интегрированной среды для Ms. Windows производится

C:\DORLAND\BIN\BCW.EXE

В результате выполнения последней команды на экране появится окно среды программирования, в верхней части которого располагается главное меню (см.рис.3.3).

Рис.3.3 - Главное меню ИСП Borland C++ 3.1

Перед тем, как начать работу по созданию и редактированию программ, следует сделать некоторые начальные установки параметров ИСП.

3.3.2. Начальные установки ИСП BC++_3.1

1. Задание полных имен основных и рабочего каталогов

Чтобы обеспечить выполнение этого действия, следует выполнить команду *Options\Directories* из меню ИСП. После этого нужно задать соответствующие имена папок в подокнах окна установки каталогов:

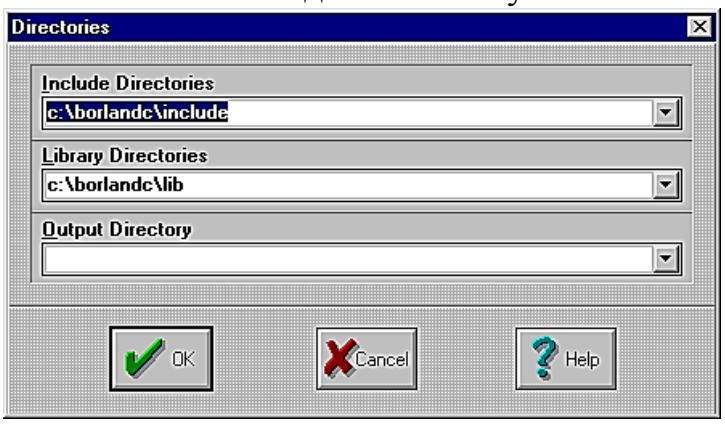


Рис.3.4 - Рабочее окно Directories

Установки в подокнах «*Include Directories*» и «*Library Directories*» требуют точного указания полного пути доступа к каталогам соответственно «*Include*» и «*Lib*» среды программирования. В этих каталогах размещаются те библиотеки со специфическими функциями, которые используются при создании программы.

Установка в подокне «*Output Directory*» –пользовательская. Она укажет компилятору среды, где ему следует сохранять редактируемые, объектные и исполняемые файлы.

Для работы в компьютерных классах факультета ТМиИ следует

- создать на диске Z каталог bcpp,

- ввести в подокно «Output Directory» строку z:\bcpp.

2. Выбор стандарта языка С.

Позволяет осуществить выбор одного из 4-х стандартов языка. Производится по команде *Options\Compiler\Source.* (см.рис.3.5)

Каждый из этих стандартов предполагает определенные правила написания программы. Например, в стандарте «*Ansi*» необходимо главную программу (в понятиях *С* – функцию) описывать так, чтобы последним ее оператором стоял оператор, возвращающий некоторый признак завершения работы. По этому признаку можно, например, узнать правильно или неправильно состоялась работа данной функции. Чтобы установить какой-либо стандарт, следует установить «флажок» слева от соответствующей надписи, используя обычные для Windows правила.

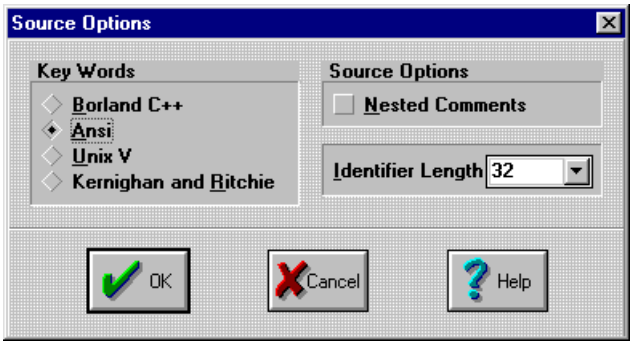


Рис.3.5 – Рабочее окно Source Options

3. Установка параметров подсистемы управления проектом (Make).

Подсистема позволяет автоматически следить за построением программы из исходных модулей на этапе ее компиляции. В случае возникновение ошибок процесс компиляции прерывается, на экран может быть выдана соответствующая ошибке информация. В случае успешного завершения компиляции в настройках подсистемы можно указать последующие действия, например, запуск компоновщика исполняемого файла и др. Настройка подсистемы производится в окне *Make* установкой флажков в соответствующих опциях. Окно вызывается в результате выполнения команды *Options\Make.*

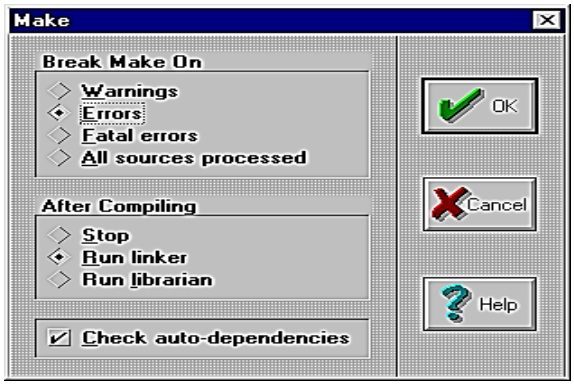


Рис.3.6 - Рабочее окно Make

4. Выбор и настройка параметров шрифтов естественного языка

По умолчанию ИСП настроена на использование одного из вариантов латинского шрифта. Поэтому для того чтобы можно было использовать тот драйвер шрифта, который используется в Вашей версии операционной системы, необходимо «подгрузить» соответствующий тип национального (русского) алфавита из набора встроенных в ИСП типов и установить его параметры. После этого клавиши переключения русского/латинского шрифтов ИСП при редактировании исходного файла программы и аналогичные клавиши Вашей версии ОС будут идентичными. Чтобы добиться этого, необходимо выполнить команду *Options\Environment\Preferences* и в подокне *Font* окна *Preferences* выбрать один из типов шрифтов с признаком **…***Cyr* (Кириллица) и его соответствующий (удобный для Вас) размер (см.рис.3.7).

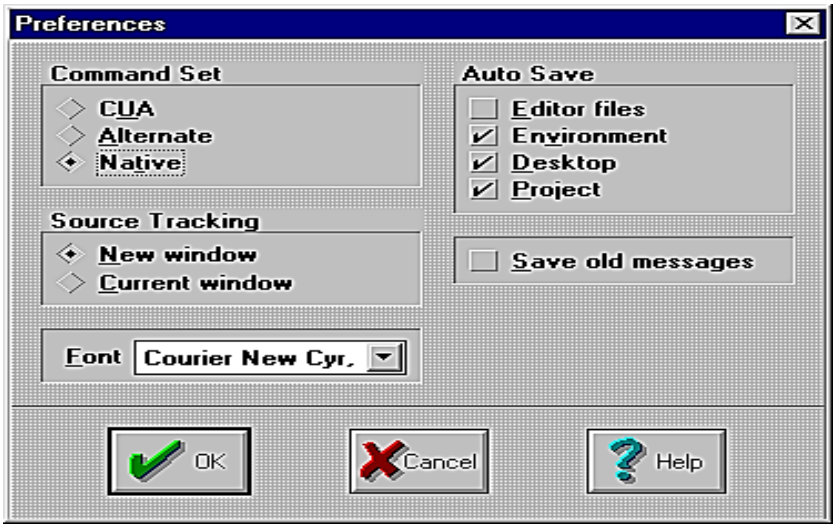


Рис.3.7 - Рабочее окно Preferences

5. Сохранение параметров настройки ИСП

Для того чтобы параметры ИСП сохранились и были доступны в следующем сеансе, необходимо выполнить команду *Options\Save.* Убедившись в том, что в опциях окна наличествуют флажки в соответствующих окнах, следует подтвердить сохранение установок нажатием кнопки «*OK*».

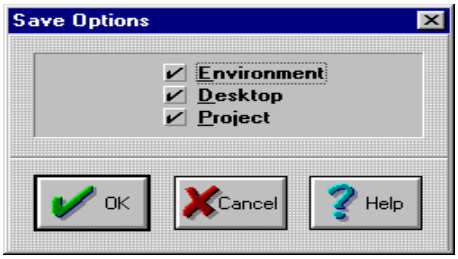


Рис.3.8 - Рабочее окно Save Option

3.3.3. Пользовательский интерфейс среды

После запуска среды программирования командой C:>\ *BORLANDC* \ *BIN* \ *BCW.EXE* на экране появиться главное меню ИСП, представленное на рис.3.3.

Рассмотрим некоторые команды этого меню.

3.3.3.1. Подсистема работы с файлами (подменю File)

Разворачивающееся меню *File* предназначено для загрузки существующих, создания новых и записи имеющихся файлов.

При загрузке файл автоматически помещается в редактор и, по умолчанию, ему приписывается безымянное имя (*noname00.cpp).* После окончания редактирования файла его можно записать в любую директорию под любым именем, для чего следует выполнить команду *FILE* / *SAVE AS* **...** В появившемся окне *Save File As* (рис. ниже). в строке *File Name* следует задать произвольное имя файла, сохранив указанное расширение. В окне *Directories* следует указать место сохранения, выбрав в списке файловой структуры диска соответствующий каталог, используя рабочие окна, показанные на рис.3.9.

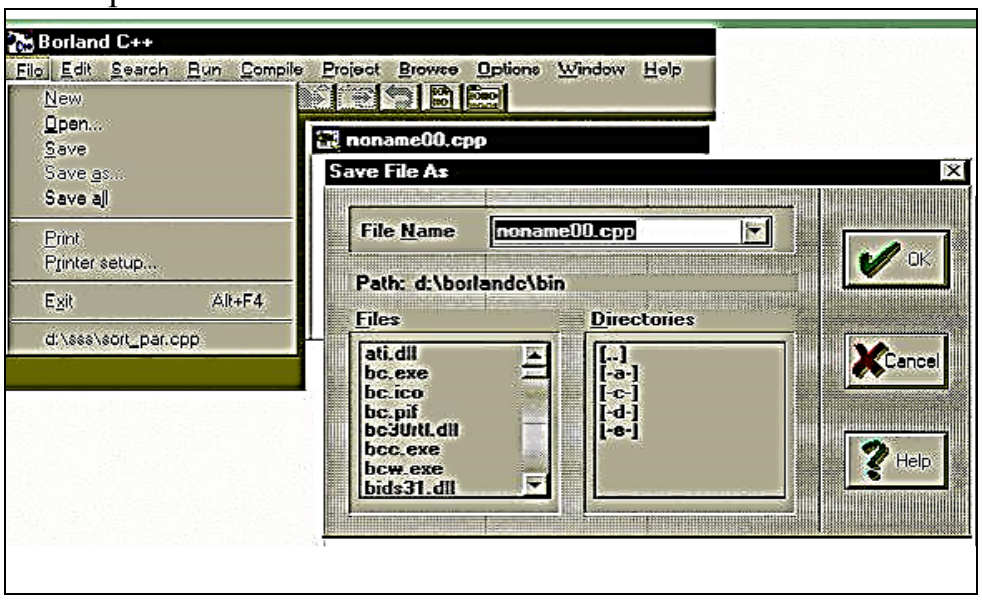


Рис.3.9 – Рабочие окна для сохранения файла

Выполняя другие команды меню *File* можно добиться следующего:

Команда *New* - указывает на то, что создается новый файл. При выборе этой команды пользователь сразу попадает в окно редактирования, которое становится активным. После набора нужного текста (программы) ее имя можно изменить на любое другое, используя команду *Save as*..

Команда *Open* позволяет находить в файловой структуре диска и открывать уже созданные файлы с исходными текстами программ.

Команда *Save* позволяет сохранять отредактированный файл под тем же именем, под которым он был сохранен ранее – переписать на диск исходный текст заново.

Задание 3.1

А) Создайте новый файл *(New).* Обычными средствами ввода данных с клавиатуры наберите в окне редактирования файла текст программы *progr1*, представленный на рисунке рис.3.2а), а затем сохраните этот файл с требуемым расширением в каталоге *bcpp* под именем *progr1.cpp*. Обратите внимание на строку имени в окне редактирования. Затем закройте данное окно редактирования.

Каталог *z:\bcpp* у вас должен быть создан заранее**.** Диск *z* – рабочий диск пользователя в компьютерных классах факультета ТМиИ университета ИТМО.

Б). Выполните команду *Open* из меню ИСП, отыщите и загрузите файл *progr1.cpp* из каталога *z:\bcpp*

Измените текст программы. Уберите строки 4,5,6, а в строке 7 напишите *c=7*.

Сохраните файл с требуемым расширением в каталоге *z:\bcpp* под именем *progr2.cpp.*

3.3.3.2. Подсистема работы с окнами

Как и все другие приложения Windows, ИСП BC_3.11 предполагает «стандартные» операции по работе с окнами. Для управления окнами, в которых одновременно может быть открыто несколько программ применяется опция главного меню *Window* (активизируйте это меню).

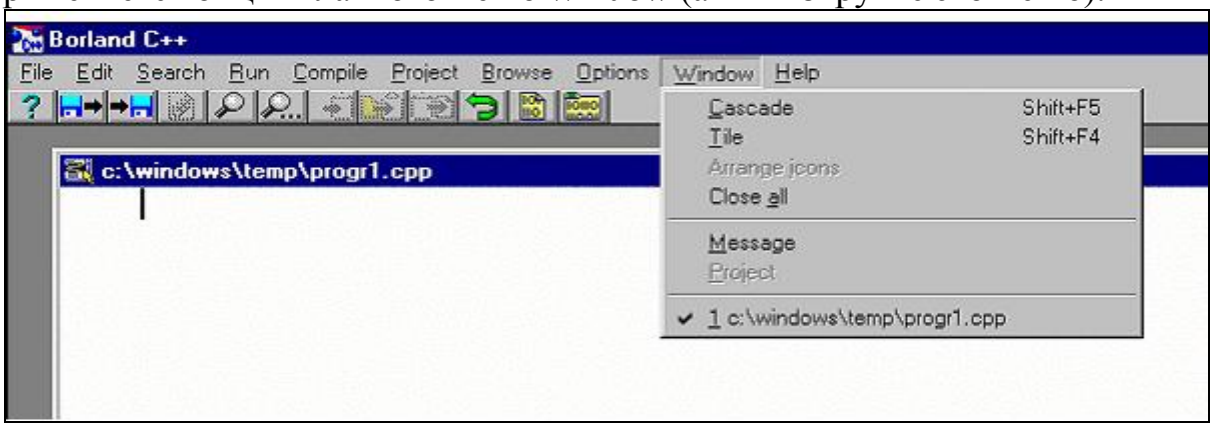


Рис.3.10 – Команды выпадающего меню Windows

С помощью данного меню можно размещать окна с разными программами так, как это удобно пользователю. Окна можно размещать:

каскадом («*Cascade*»),

рядом друг с другом («*Tile*»),

в виде иконок(«*Arrange icons*»), скрывать(«*Close all*»);

разворачивать (активизировать, установив флажок напротив названия программы в нижней части всплывающего меню).

Кроме того, на экране можно размещать дополнительные окна, такие, как окно сообщения об ошибках («*Message*») или окно, показывающее файлы, входящие в проект программы («*Project*»). Все операции по переходу

из одного окна в другое – стандартны для ОС. Здесь только следует помнить о том, что дополнительные окна (*«Message», «Project»)* сопутствуют только тем окнам с программами, которые активны в данный момент.

Задание 3.2

Откройте два созданных исходных файла. Вначале выберите команду *Tile* или *Cascade,* и разместите окна с этими программами так, как вам удобно.

Затем, выбрав команду *Message,* разместите на экране окно сообщений об ошибках. Обычно это окно располагают ниже окон с программами. Зафиксируйте факт о том, что данное окно соответствует только одному (активному) окну с файлом программы.

3.3.3.3 Работа по редактированию файлов программ

Редактирование программ достигается либо привычными стандартными средствами редактирования, такими, как работа над элементами текста с использованием манипулятора мышь, либо с помощью использования операций, предоставляемых меню *Edit.*

Данное меню позволяет

- ✓ отменять(«*Undo*»),
- ✓ или возобновлять(«*Redo»)* действие по редактированию текста,
- ✓ вырезать в «карман» (буфер обмена) фрагмент текста(«*Cut»),*
- ✓ копировать фрагмент(«*Copy*»),
- ✓ вставлять его из «кармана»(«*Paste*»),
- ✓ полностью очищать окно с программой или очищать (удалять) ее фрагмент («*Clear»).*

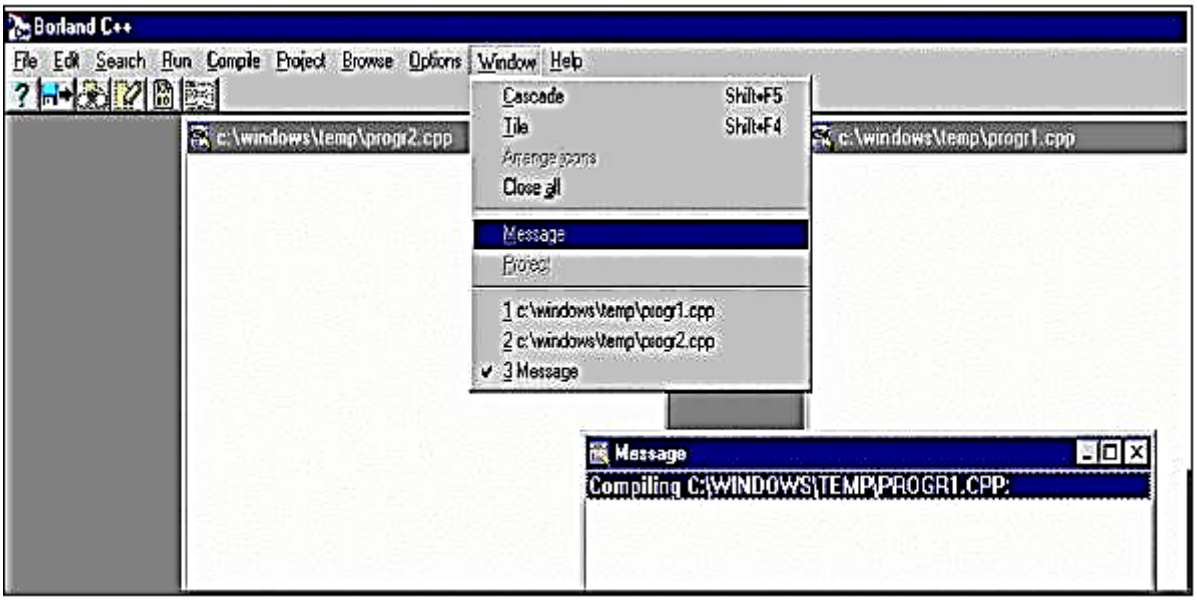


Рис 3.11 - Работа с окнами

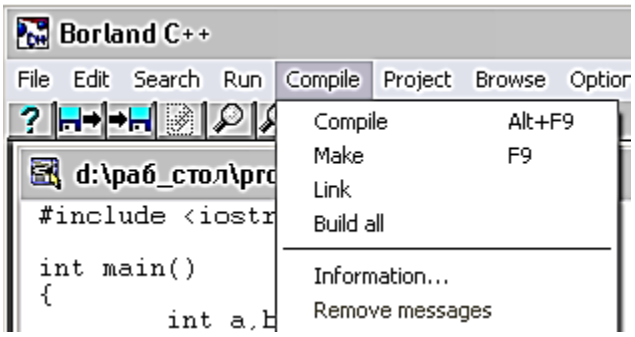
Выделение редактируемого фрагмента достигается стандартными манипуляциями с мышью или клавиатурой, а операции над ним, помимо стандартных средств ОС, осуществляется либо при помощи выбора

соответствующей опции данного меню, либо используя «горячие клавиши», которые указаны в надписях меню правее его опций.

3.3.3.4 Работа с файлами при компиляции программ

Процесс создания выполняемой программы в среде Си можно представить в виде простой схемы:

progr1.cpp → progr1.obj. →progr1.exe



Компиляция и компоновка программы после ввода текста осуществляются с помощью раскрывающегося меню *Compile*.

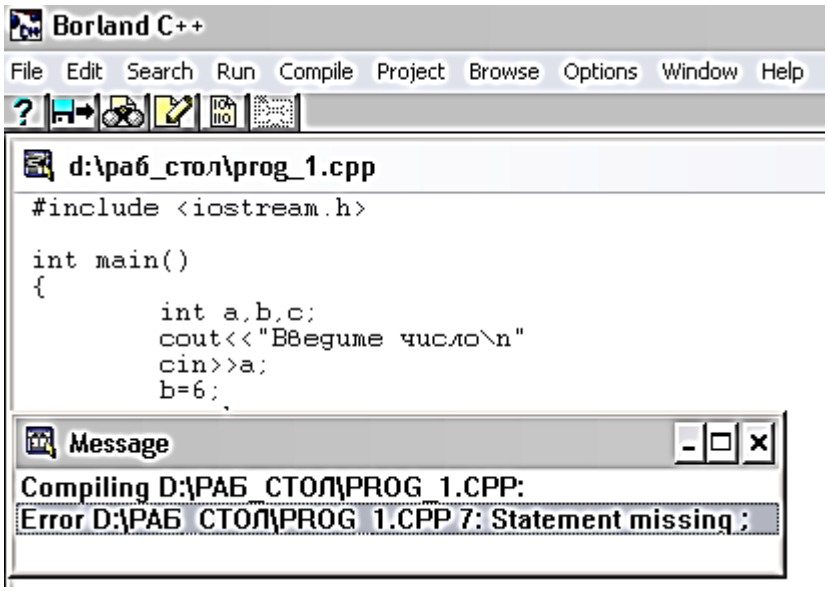
Рис. 3.12 - Команды меню Compile

Команды этого меню:

- ✓ *Compile* – команда трансляции исходного текста программы в объектный файл.
- ✓ *Make* – команда построения исполняемого файла.
- ✓ *Link* – команда компоновки текущего объектного файла с внешними библиотеками С (*LIB).
- ✓ *Build all* – команда перетрансляции всех файлов, входящих в проект, независимо от даты их *Information* – команда выдачи статистических данных результатов компиляции.
- ✓ *Remove message* – команда обновления сообщения об ошибках.

Задание 3.3

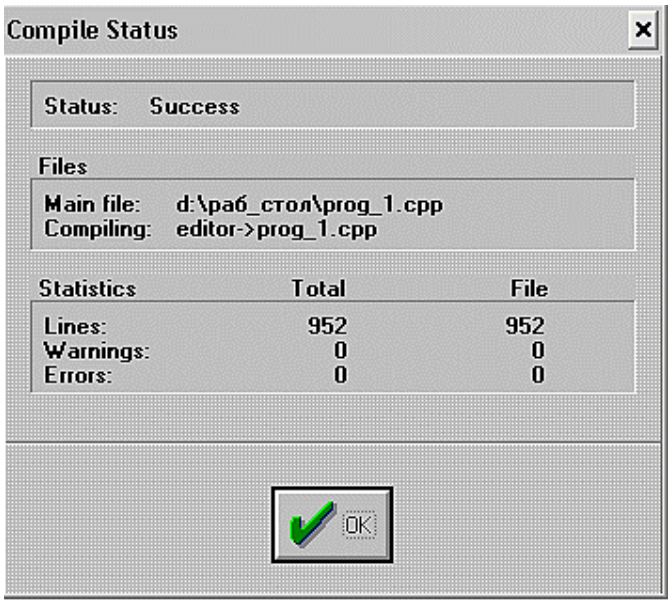
Активизируйте окно с файлом *progr1.cpp*. - первой программы.



Принудительно введите ошибку, например, удалите точку с запятой в строке с оператором ***cout.***

Выполните команду *Compile.* Проследите за результатом компиляции и содержимым окна сообщений об ошибках.

Рис.3.13 - Результат компиляции файла с ошибкой



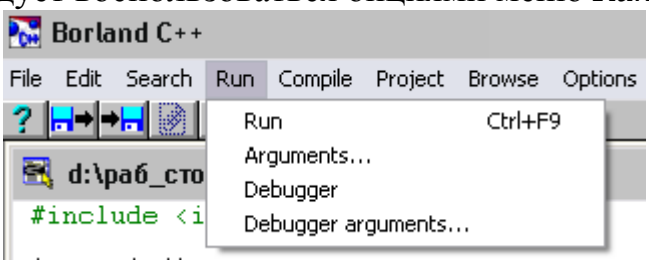
Устраните ошибку и вновь выполните команду *Compile*. Проанализируйте результат компиляции, обратив внимание на окна статуса *Status* и статистики *Statistics*.

Рис.3.14 - Рабочее окно Compile Status

3.3.3.5. Работа с файлами при выполнении программ

После набора текста программы в окне редактора и ее успешной компиляции появляется возможность ее выполнения и просмотра результатов ее работы. Трансляция, компоновка, загрузка и выполнение могут быть реализованы сразу, с помощью выполнения одной лишь команды ИСП. Но рекомендуется разделять эти процессы.

Чтобы запустить на выполнение отлаженную программу, которая не имеет ошибок, следует воспользоваться опциями меню *Run*.



Команда «*Run*» этого меню позволяет автоматически выполнить исполняемый файл, сформированный после компиляции и линковки. Команда «*Debuger*» позволяет совершить пошаговое (построчное) выполнение инструкций программы. Эта команда полезна в тех случаях, когда на этапе компиляции ошибки не выявлены, а результат выполнения программы при этом является некорректным. Так происходит, как правило, в результате нарушений в логике алгоритма программы.

Задание 3.4

Активизируйте окно с файлом успешно откомпилированной первой программы.

Выполните команду *Run*.

Проследите за результатом выполнения программы. Он показан на рис 3.2б).

3.4 ИНТЕГРИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ VISUAL C++ 2005-2012-…

Программирование на универсальном языке *Visual C++ 2005-2012* происходит в *интегрированной среде программирования (ИСП) - (Integrand Development Environment (IDE)*, которая является одной из подсред пакета *Visual Studio.* Ее структура и интерфейс не представляют сложности в общении и построены традиционным образом: окна, меню, панели инструментов и т.д. Отдельные особенности *ИСП* (или *IDE* в оригинальном представлении) рассмотрим по мере изучения материала.

3.4.1. Структура программ в VC++ и начало создания проекта

Вопросы, касающиеся структуры программ, отчасти были рассмотрены выше. Здесь сделаем упор на программы в среде *VC++ 2005* и других более высоких версий. Программы в *VC++* называются приложениями. Приложения строятся средой в виде отдельных конструкций — проектов, которые представляют из себя совокупность нескольких файлов. Любое приложение включает главную функцию, внутри которой помещаются операторы, реализующие алгоритм приложения. Среди операторов могут содержаться и такие, которые вызывают другие функции, включенные в приложение. Эти функции либо создаются программистами, либо являются частью среды программирования.

Изучая программирование на начальном этапе, используют обычно *консольные приложения.* Для них в среде *ИСП* заготовлены специальные шаблоны, которые упрощают создание программ. *Консольными приложениями* называют такие, которые запускаются пользователем через специальную командную строку или командой IDE. При этом *консолью называется: при вводе – клавиатура, при выводе - дисплей*. Консольные приложения создают, используя шаблон, запуск которого производится после выполнении команды меню *IDE - File\New\Project* из рабочего окна, вид которого представлен на рис.3.15.

В окне *New Project* следует выбрать *тип проекта (Project types*) - *CRL* и *шаблон (Templates) - CRL Console Application.* Кроме этого, в поле *Name* следует дать имя проекту (у нас это *PVU_00*), а в поле ввода *Location* определить папку для хранения этого проекта.

CLR (Common Language RunTime) представляет собой специальную среду, которая управляет исполнением программного кода, памятью, потоками данных и работой с удаленными компьютерами, при этом строго обеспечивая надежность исполнения кода. Такое приложение отличается от обычного тем, что его заготовка обеспечивает подключение к приложению специального системного пространства *System,* содержащего объекты, размещение в памяти, которые надо автоматически регулировать. *CLR*

работает с управляемой памятью, в которой размещение объектов и ее освобождение от них происходит под управлением среды IDE [13].

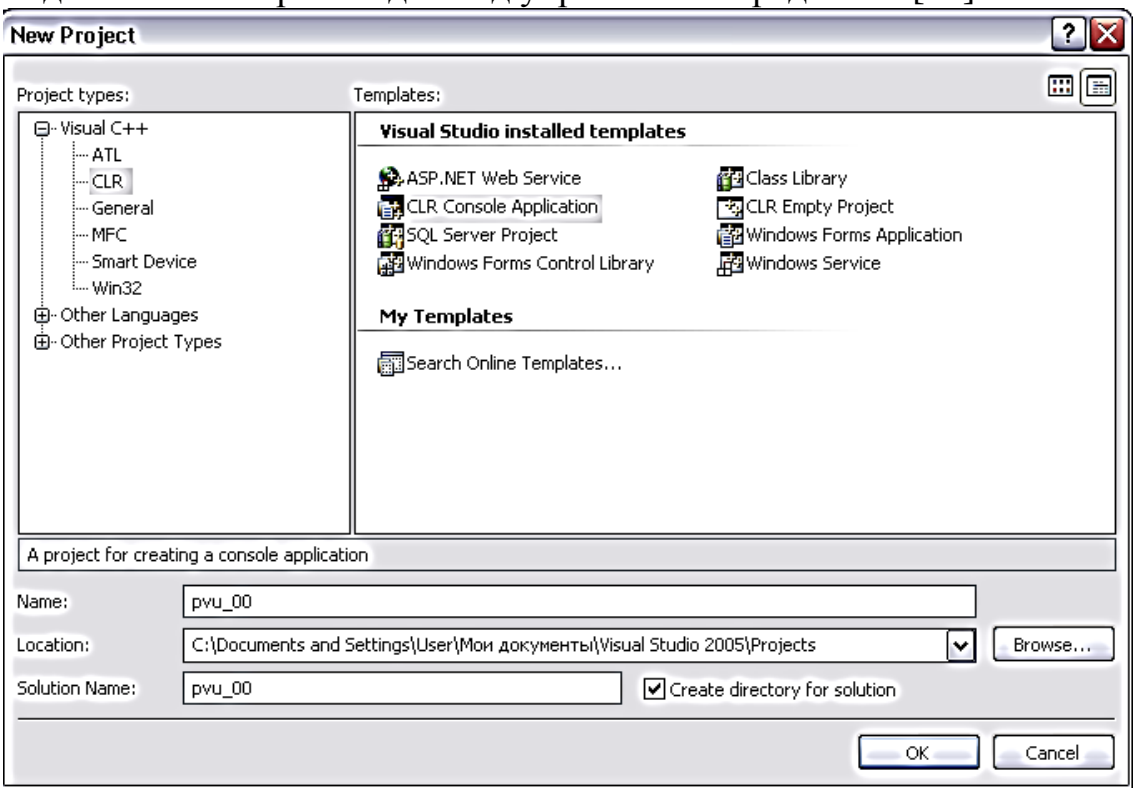


Рис.3.15 - Вид окна New Project

3.4.2. Создание простейшего консольного приложения

После щелчка по кнопке *OK* в окне *New Project* (см.рис.3.15) на экране появится экранная форма *pvu_00 –Microsoft Visual Studio*, изображенная на рис.3.16.

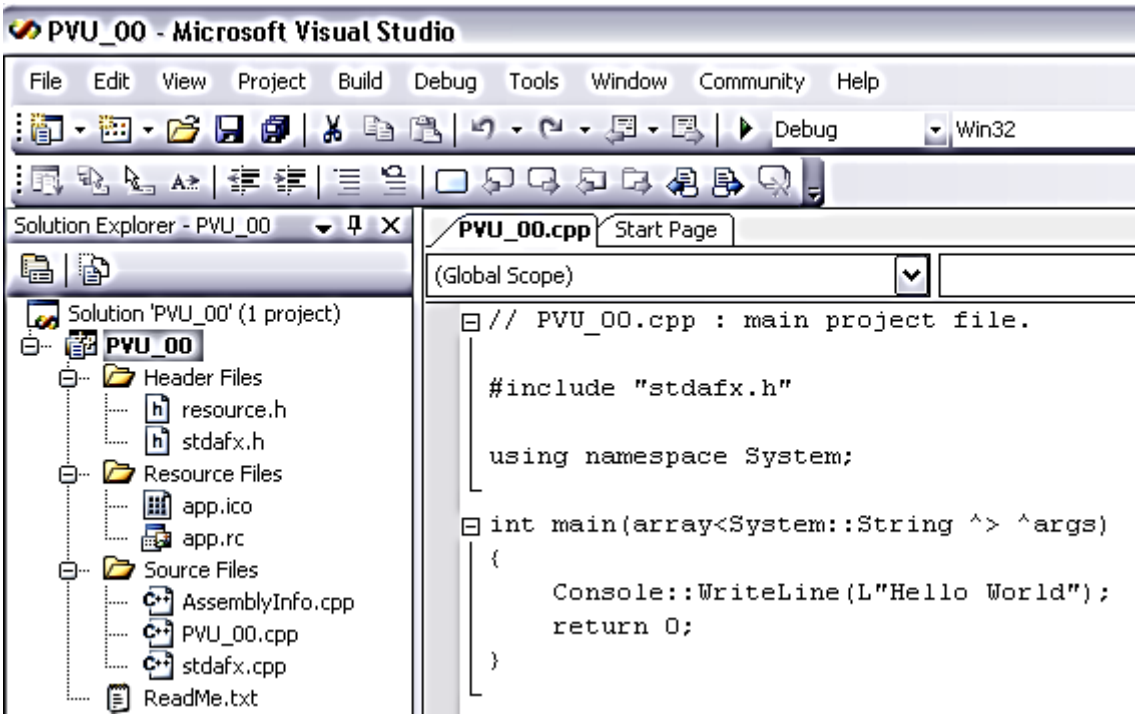


Рис.3.16 - Шаблон консольного приложения

На форме видны меню, панели инструментов и два окна. Первое окно– *Solution Explorer (группа проектов, хотя Solution переводиться как решение),* содержит файлы, входящие в проект, и второе - с двумя вкладками. Первая вкладка, повторяющая заголовок проекта *PVU_00.cpp,* включает шаблон консольного приложения, вторая вкладка *Start Page* содержит стартовую страницу среды.

После появления шаблона проекта на экране в него добавляют необходимые спецификации, операторы и функции. Компиляцию производят командой *Build,* которая появляется в меню после создания проекта. Запуск проекта на выполнение осуществляют командами меню опции *Debug.*

Некоторые файлы из проекта рассмотрены ниже.

PVU_00.cpp— главный исходный файл и точка входа в создаваемое приложение (*PVU_000* — это имя проекта). Этот файл рассмотрим подробнее ниже.

stdafx.cpp — файл, который подключает для компиляции файл *stdaf.h.*

В файле *stdafx.h* пользователем задаются дополнительные файлы оглавления, если они требуются для проекта.

ReadMe.txt— файл, описывающий некоторые из созданных по шаблону консольного приложения файлов проекта.

Посмотреть содержимое указанных файлов можно через их контекстные меню, если просто щелкнуть мышью на имени файла. В этом случае в окне справа от окна *Solution Explorer* появляется содержимое файла, а над содержимым — вкладка с именем открываемого файла.

Создадим новый проект с именем 567 и рассмотрим главный исходный файл PVU_00.cpp. Шаблон его консольного приложения будет повторять шаблон, представленный на рис.3.16.

Инструкция *using namespace System* обеспечивает подключение к приложению специального системного пространства *System* и использование соответствующих имен. Без этой инструкции невозможно выполнение оператора, который выводит на экран слова «*Hello World»*. В дальнейшем в данном пособии команда *Console::WriteLine(L"Hello World*") использоваться не будет. Для этого существует большое число других операторов языка.

Заголовок главной функции, помимо названия *main,* включает формальные параметры, заключенные в круглые скобки. Удалим их и перейдем к функции без аргументов, как показано на рис.3.17.

Добавим в шаблон некоторые инструкции (см.рис.3.17.).

Инструкция //02 -*#include* <*iostream*> подключает к программе библиотечные средства консольного ввода-вывода.

Инструкция //03 - *using namespace std* –позволяет обращаться к пространству имен стандартной библиотеке языка С++, содержащей операторы управления потоками ввода-вывода.

Если не подключить инструкцию *//03*, то команда *//031* выполнятся не будет.

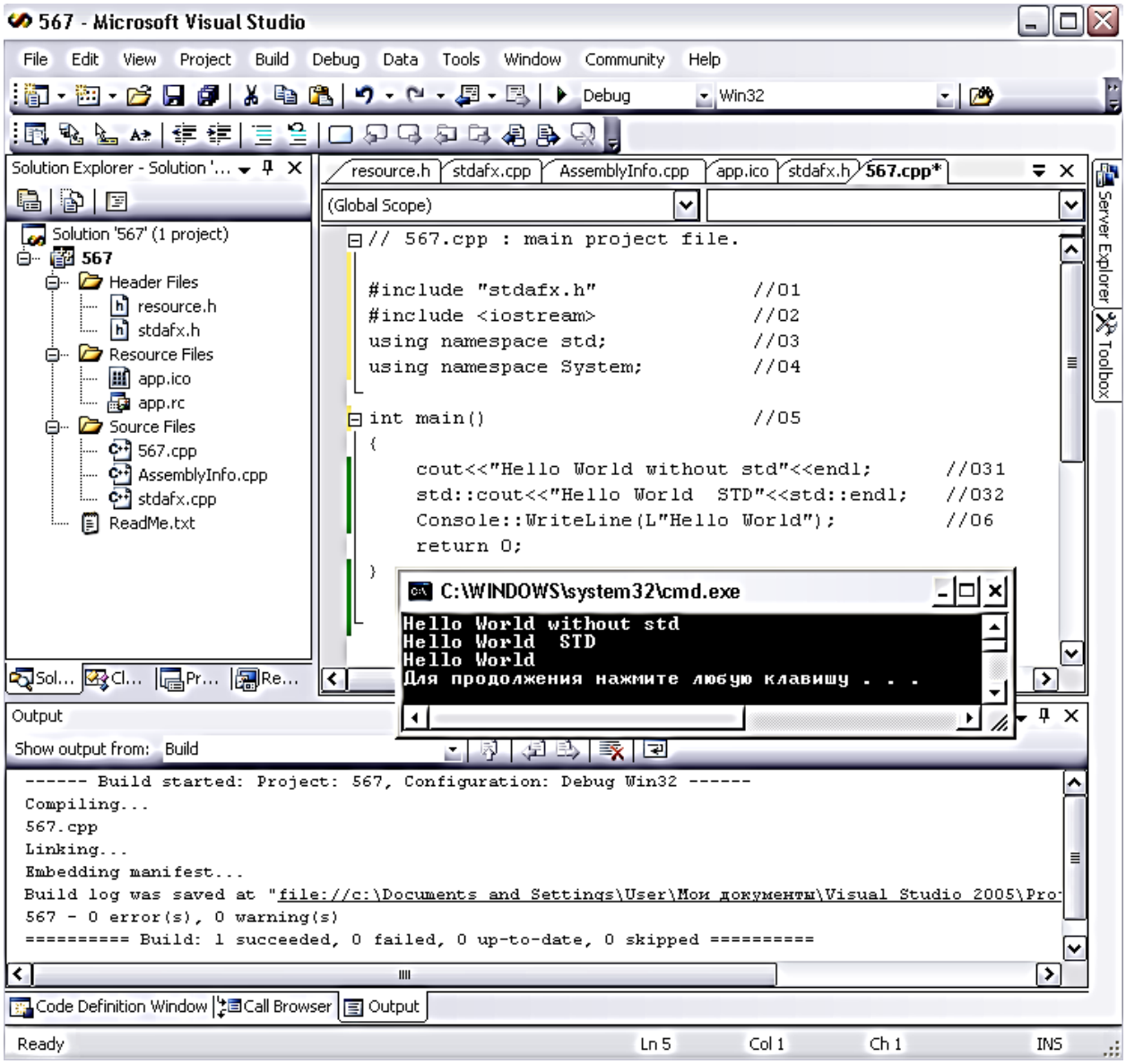


Рис.3.17 - Исправленный шаблон консольного приложения

Инструкция //01 *-#include <stdafx.h>* должна первой в списке инструкций в разделе деклараций. Если поменять местами инструкции //01 и//02 компилятор может выдать ошибку.

При вводе и выводе информации и некоторых других действиях в новых версия *VC++* используются конструкции операторов и инструкций, каких не было при работе с компилятором *Borland C 3.1*. Это можно заметить, сравнивая приведенные варианты программ с программами, написанными для компилятора *Borland C 3.1*, рассмотренными ранее. Некоторые замечания в этом плане можно отыскать у Подбельского В.В. [14].

До введения новой версии международного стандарта ISO/IEC 14882, принятой в 2003 г.[14], обозначения заголовочных файлов программ имели

расширение *".h"*. Таким образом, устаревший вариант препроцессорной директивы подключения к программе библиотечных средств консольного ввода-вывода был таким: *#include <iostream.h>*.

Первое изменение (удаление суффикса *".h"* из названия заголовка) - это принятое в Стандарте соглашение. *"std"* — принятое обозначение пространства имен стандартной библиотеки языка *C++*. Лексема **::** — это операция указания области видимости.

Второе отличие стандартной программы от ее устаревшего варианта - префикс *"std::"* в именах стандартного выходного потока *std::cout* и манипулятора *stdr.endl.*

Пространство имен — это средство, позволяющее группировать и локализовать обозначения, использованные в библиотеке или программе. Если пространство имен при разработке программы не указано, то предполагается, что все глобальные имена (например, имя *main* главной функции программы), использованные в программе, находятся в единственном глобальном пространстве имен. Поэтому ранее при использовании нестандартного заголовка <*iostream.h*> имена *cout* и *endl* в программах употреблялись без префиксов.

Стандарт поместил имена из стандартной библиотеки классов и функций в пространство имен *std,* т.е. отделил эти имена от глобального пространства. Теперь к именам библиотечных средств можно обратиться, указав, что разыскивать их нужно в пространстве имен *std* (а не в глобальном пространстве имен).

Так как при частых обращениях к библиотеке довольно обременительно снабжать каждое имя библиотечного средства префиксом *std::,* то возможно применение в программе специального описания такого вида:

using namespace имя пространства _имен;

Это описание позволяет обращаться к именам названного в нем пространства с помощью неполного имени, не содержащего префикса. Описание

using namespace std;

делает доступными имена из *std* в той части программы, перед которой это описание размещено.

Старые компиляторы не справятся с обработкой этой программы. Если у вас такой компилятор, как, например, *Borland C++ 3.0*, обойти это затруднение несложно - замените в программе *567.cpp* строки

#include <iostream>

using namespace std;

на одну строку *#include <iostream.h>*

Теперь ее будет компилировать и устаревший компилятор.

Разработчики современных компиляторов с целью сохранения преемственности встраивают в них возможность применения в программах старых заголовков (с расширением. *h.).* Но это уже не относится к Стандарту.

Компиляция программы осуществляется командами меню *Build Solution* или *Rebuild Solution*, показанными на рис.3.18.

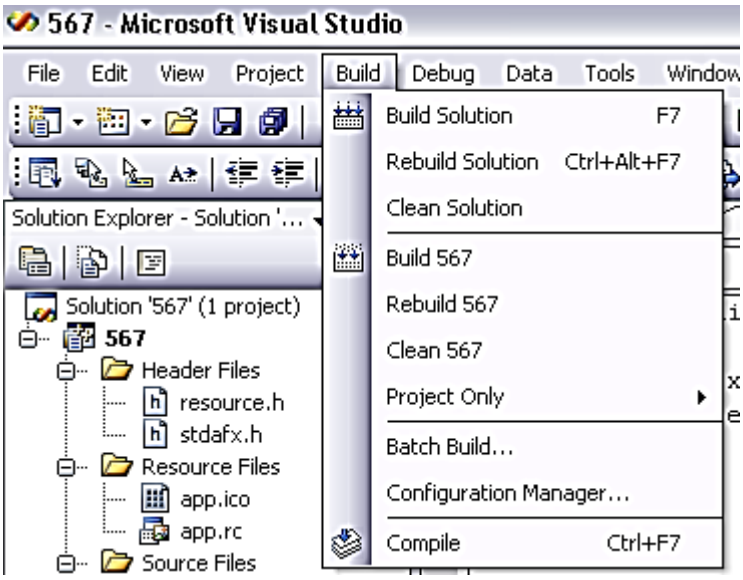


Рис.3.18 - Инструкции меню команды Build

В результате выполнения этой команды в нижнем окне рисунка (см. рис. 3.17) будут размещены сведения об ошибках и других результатах выполнения программы. Если ошибок нет, то можно запустить программу на выполнение.

Выполняется программа инструкцией *Start Debugging* или *Start Without Debugging (см.рис.3.19.)*

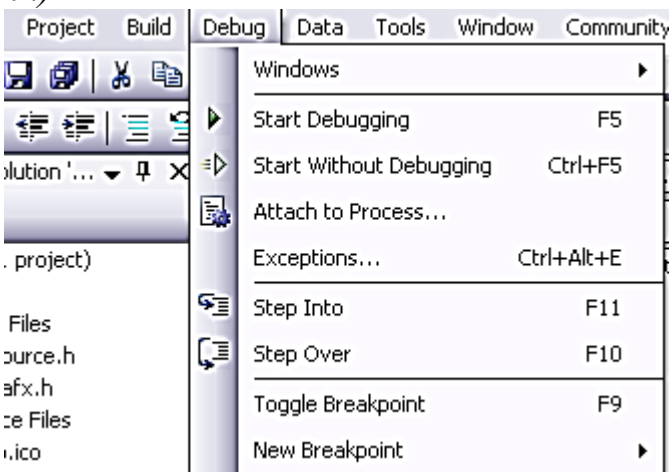


Рис.3.19 - Инструкции команды меню Debug

Результат выполнения этой команды (программы) представлен на рис.317. в черно-белом окне *C:\Windows\System32\cmd.exe* /

Задание 3.5. Создание первого консольного приложения в *VC++ 2008*

В качестве первой программы возьмем программу, которую исследовали и создали ранее (см. рис.3.2). С некоторыми изменениями в соответствии с рассмотренным материалом программа будет иметь вид, представленный на рис.3.20, ниже.

На рис.3.20 Б) инструкция *using namespace std* представлена как комментарии, т.е. не работает. Это приводит к тому, что для вывода и ввода информации в соответствующие потоки следует использовать перед именами потоков вставку *std::*.

Кроме того, в этих программах помимо управляющего символа -*n* использован манипулятор *endl*, выполняющий те же функции.

Заметим, что поясняющий текст в программе -*"Vvedi chislo "* набран на латинице. Русский текст компилятор не распознает. Для того, чтобы компьютер понимал русский текст необходимо разместить в разделе деклараций препроцессорную директиву:

#include "cyrToDos.h"

```
// 567.cpp : main project file.

#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{    int  a, b, c;
     cout<<"Vvedi chislo\n";
     cin>>a;
     cout<<"a="<<a<<endl;
     b=6;
     c=a+b;
     cout<<"c="<<c<<endl;
}
```

А)

```
// 567.cpp : main project file.

#include "stdafx.h"
#include <iostream>
//using namespace std;

int main()
{    int  a, b, c;
     std::cout<<"Vvedi chislo\n";
     std::cin>>a;
     std::cout<<"a="<<a<<std::endl;
     b=6;
     c=a+b;
     std::cout<<"c="<<c<<std::endl;
}
```

Б)

Рис.3.20 - Первая программа на VC++ 2008

С ее помощью в текст программы может быть добавлен код из файла *cyrToDos.h,* текст которого размещен в книге В.В.Подбельского [14].

Добейтесь выполнения обеих программ. Сохраните проекты.

3.5. ЛЕКСИЧЕСКИЕ ОСНОВЫ С++

3.5.1. Типы данных

Любая программа в процессе своего выполнения оперирует с данными. Эти данные могут быть различных типов. Типы данных, которые использует С++ представлены на рисунке рис. 3.21.

К простым типам данных относят целые числа, вещественные числа и символы. В программе тип данных должен быть указан так, как показано в таблице 3.2.

Тип показывает сколько памяти отводится для данных при компиляции. Этот размер зависит от типа используемой ПЭВМ. Если,

например, машинное слово 2 байта (16 бит), то под int отводится 2 байта, под short – 2 байта, под long – 4 байта, под– 2 байта.

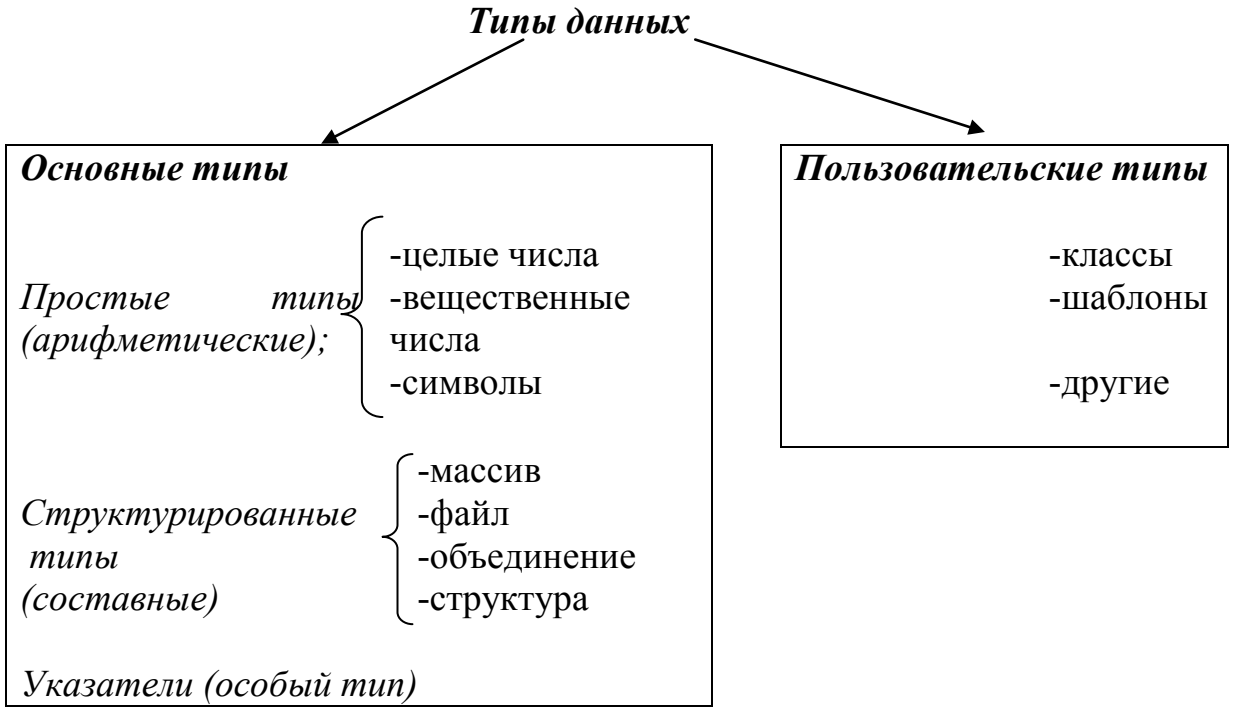


Рис. 3.21. - Типы данных

Таблица. 3.2- Указание типов переменных

Целочисленные типы:	***Размер байт/(бит)***	***Диапазон значений***
int – целый;	2/(16)	-32 768 - 32 767
short int (short) – короткий целый;	2/(16)	-32 768 - 32 767
long – длинный целый;	4/(32)	-2 147 483 648 - 2 147 483 647
unsigned – беззнаковое целое;		
unsigned int – целый б/з;	2/(16)	
unsigned short – короткий целый б/з;	2/(16)	
unsigned long – длинный целый б/з;	4/(32	0…..65 535 0…..65 535 0 4 294 967 295
Вещественные типы:		
float – одинарной точности;	4/(32)	-3.4E-38….3.4E+38
double – удвоенной точности;	8/(64)	-1.7E-308….1.7E+308
long double – максимальной точности	10/(80)	-3.4E-4932…3.4E+4932
Символьный тип		
char	1/(8)	-128….+127
unsigned char	1/(8)	0…255

Под *float* обычно отводится в два раза больше памяти чем под *int*. Под данные, описанные как *double*, отводится в два раза больше памяти чем под *float*.

Символы представляются типом *char*, под который отводится 1 байт.

Заметим *A1).* К переменным целого типа относят логические переменные

bool – булевская переменная, принимающая 2 значения – *true и false.*

A2). enum –перечисляемый тип.

Переменным, перечисленным в списке, присваиваются

- ✓ последовательно целые значения, начиная с нуля,
- ✓ заданные значения при явном присвоении.

Например:

enum { Black, Blue} ; // Black =0, Blue=1

enum qaz { ww=111, ss=222, xx=333 };

//qaz может принимать только значения, указанные в списке.

A3). Для корректного выполнения программы каждая переменная может использоваться для записи и хранения строго определенных типов данных. Например, если создана переменная для обработки целых чисел, то символ или вещественное число этой переменной уже присвоить нельзя.

A4). Перед использованием переменной в программе ее необходимо заблаговременно объявить. Процедура объявления переменной предполагает, во-первых, указание типа этой переменной и, во-вторых, создание отличного от любого ключевого слова идентификатора (имени переменной).

Примеры объявления переменных:

int a; // объявлена целочисленная переменная с именем *a*

float a1, f; // объявлены две вещественные переменные *a1* и *f.*

unsigned int year=2009 ; // инициализация переменной

A5). На физическом уровне, объявление переменных означает, что в области оперативной памяти компьютера выделяется именованный участок памяти определенного размера, который соответствует указанному типу данных и к которому можно обратиться по имени переменной для записи или считывания данных.

A6). Все описанные переменные в программах должны принимать какие-либо значения. Эти значения должны *присваиваться* им явно при объявлении с помощью оператора присваивания (=), например, *int a1=5,* или вычисляться в ходе выполнения программы, например, *float a2 = sqrt(a1).*

3.5.2 Константы

Неизменяемые величины называются константы. Это числа или символы, используемые в программе. Числа могут представляться в различных системах счисления.

Целые числа. Эти константы представляются типом *int.*

- *Десятичная константа* изображается цифрами от 0 до 9. Первая цифра не может быть нулем.
-
- *Восьмеричная* содержит цифры от 0 до 7, но первая цифра обязательно 0 (нуль).

Пример: $015 \rightarrow 13_{(10)}$,
$052 \rightarrow 42_{(10)}$.

Шестнадцатеричная константа начинается с комбинации символов *0x* или *0X* (нуль, икс).

Пример: $0x10 \rightarrow 16_{(10)}$,
$0X25 \rightarrow 37_{(10)}$.

Константы с плавающей точкой представляются типом *double* и записываются в виде мантиссы и порядка.

Пример: $113.25e\text{-}2 \rightarrow 113{,}25 \circ 10^{-2} \rightarrow 1{,}1325_{(10)}$,
$3.7E25 \rightarrow 3{,}7 \circ 10^{25.}$

Символьная константа состоит из одного символа (буква, цифра, специальный символ), заключенного в апострофы.

Пример: char SIM;
SIM='A';
char a='5';

Строку символов также относят к константам (строковым). Для их представления используются двойные кавычки:

Пример: "Error", "125", "d".

При записи строковой константы в память, компилятор в ее конец помещает символ *'\0'(нуль-терминатор*), отмечающий конец строки.

Управляющие константы. Для этого, чтобы их отличать от простых констант, они используются в связке с символом '\' (обратный слэш).

Пример: \n – перевод курсора в начало следующей строки (вертикальная табуляция),
\t – знак горизонтальной табуляции.
\\ - обратная наклонная черта,
\" - двойная кавычка

3.5.3. Понятие массива

Массив относится к составному типу данных. Он представляет собой набор простых *однотипных* данных. Они расположены в памяти вплотную друг за другом. При объявлении массива, после указания типа, записывают имя массива, а за ним в квадратных скобках указывается количество элементов массива.

Объявляют массивы так:

int array[10]; //Целочисленный массив с именем *array* из 10 элементов.

float mass[23]; //Массив из 23 вещественных чисел. Имя массива *mass*.

char S[25]; //Массив *S* из 25 символов.

При использовании массива в программе число в скобках указывает *номер элемента массива* и называется *индексом*. *Нумерация индексов* в С/С++ всегда *начинается с нуля.*

Примеры.

1) *mass[2] = 4.13;* //Третьему элементу массива с именем *mass* присвоено значение 4.13.

2) *j = 1;*

array[j] = 8; //Второму элементу массива *array* присвоено значение 8.

3.5.4. Имена переменных

Программа производит вычисления. Для вычисления значений используются *выражения,* состоящие из операндов, знаков операций и скобок. *Операнды* задают данные для вычислений. *Операции* задают действия, которые приходится выполнить. Каждый операнд может быть является выражением.

Переменная- это именованная область памяти, в которой хранятся данные определенного типа. У переменной есть *имя* – для обращения к области памяти, в которой хранится ее *значение*. Перед использованием любая переменная должна быть описана.

Алфавит (или множество литер) языка программирования С++ основывается на множестве символов таблицы кодов ASCII. Алфавит С++ включает:

- строчные и прописные буквы латинского алфавита (мы их будем называть буквами),
- цифры от 0 до 9 (назовём их буквами-цифрами),
- символ '_' (подчерк - также считается буквой),
- набор специальных символов:

  " { } , | [ ] + - % / \ ; ' : ? < > = ! & # ~ ^ . *
- прочие символы.

Алфавит С++ служит для построения слов, которые в С++ называются *лексемами.* Различают следующие *типы лексем*:

- *идентификаторы* (для задания имен константам и переменным),
- *ключевые слова,*
- *знаки (символы) операци.*

Ключевые слова

Часть идентификаторов С++ входит в фиксированный словарь ключевых слов. Это такие как: *asm auto break case catch char class const continue default do double else enum extern float for friend goto if inline int long*

new operator private protected public register return short signed sizeof static struct switch template this throw try typedef typeid union unsigned virtual void volatile while.

Правила образования идентификаторов

1. Первым символом идентификатора C++ может быть только буква.
2. Следующими символами идентификатора могут быть буквы, цифры символ подчеркивания.
3. Длина идентификатора не ограничена (фактически же длина зависит от реализации системы программирования).
4. Идентификатор не должен совпадать с ключевыми словами, т.е. словами, которые уже зарезервированы и используются в ЯП.

3.5.5. Описание переменных и область их действия

Общий вид описания переменных:
[класс памяти] [const] тип имя [инициализатор]

Необязательный модификатор *const* показывает, что такую переменную изменить нельзя. Она называется *константой*.

Инициализатор. Присвоение переменной значения при ее описании называют инициализацией. Инициализация происходит в двух формах:

- со знаком равенства - *char c='f' или const char c='f'*
- *в круглых скобках char c ('f')*

Необязательный *класс памяти* может принимать одно из значений:
auto, extern, static, register

Класс памяти и место описания переменной в тексте программы определяют видимость переменной и время ее жизни. *Область действия идентификатора* – это часть программы, в которой его можно использовать для доступа к связанной с ним области памяти

Блок – часть программы, ограниченная фигурными скобками.

Если переменная определена внутри блока, она называется *локальной*. Область ее действия распространяется от точки описания блока до конца блока, включая все вложенные блоки. Если переменная определена вне любого блока, она называется *глобальной*, и областью ее действия считается файл, в котором она определена, от точки описания до конца файла.

Время жизни может быть постоянным (в течение всего выполнения программы) и временным (в течении выполнения блока).

Областью видимости идентификатора называется часть программы, из которой допустим обычный доступ к связанной с идентификатором область памяти. Чаще всего область видимости совпадает с область действия.

Исключением является ситуация, когда во вложенном блоке описана переменная с тем же именем. В этом случае переменная, которая описана вне

вложенного блока, во вложенном блоке не видна, хотя она входит в область ее действия.

Но если эта переменная глобальная к ней можно обратиться, используя операцию доступа к области видимости **::**

Классы памяти:

auto- автоматическая переменная

Для глобальных переменных не используется. Для локальных принимается по умолчанию, поэтому задавать явным образом смысла нет.

Рождается и умирает при входе и выходе из блока. Память каждый раз освобождается.

extern – внешняя переменная

Определяется в другом месте программы (в другом файле, далее по тексту).

Используется для создания переменных, доступных во всех модулях программы, в которых они объявлены.

static - статическая переменная

Время жизни –постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статическая переменная может быть глобальной или локальной.

register – регистровая переменная

Переменная целого типа, располагающаяся в регистрах компьютера. Если такую возможность компилятор не отыщет, то переменная обрабатывается как *auto*.

Задание 3.6

Пример программы с различными описаниями переменных и результатом ее выполнения приведен на рис.3.22. Эта программа написана для компилятора *Borland C++ 3.*

Объявление переменных обязательно. Программа содержит *комментарии*. Если комментарии занимают одну строку, используем двойной слеш. Если несколько строк - заключаем их в символы /*..*/.

Ваша задача - добиться работоспособности программы, используя средства *VC++ 2005* или более высокой версии. Необходимо уметь объяснить, как работают все инструкции и команды программы. Подумайте, что будет если исправить отдельные команды или убрать их совсем. Попробуйте поэкспериментировать с программой и объяснить результаты ее работы в результате произведенных действий.

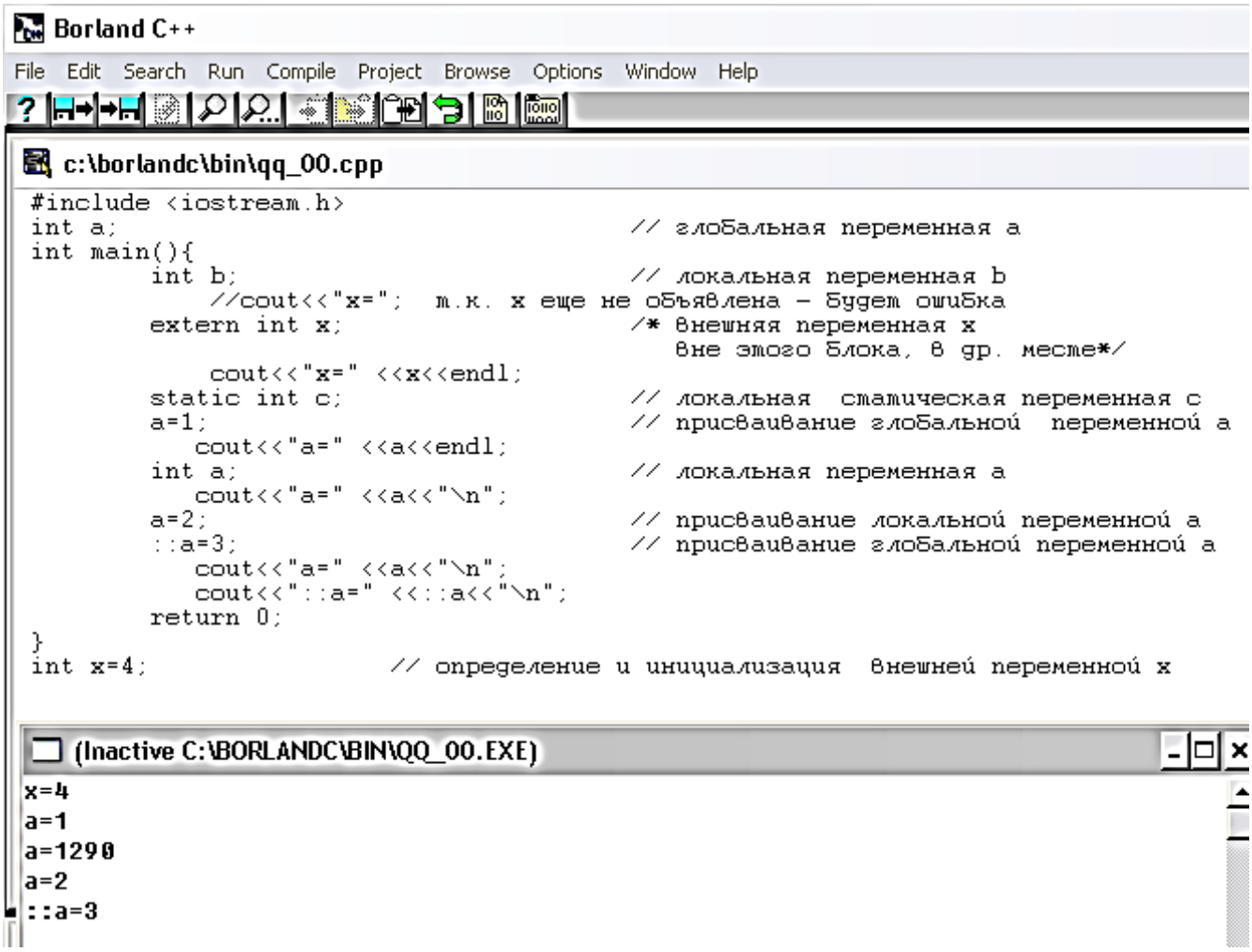


Рис.3.22 - Описание переменных

3.5.5. Операторы языка С

Оператор — это запись в виде определенных символов той или иной операции. Их делят на операторы преобразования данных и операторы управления работой программы. Оператор определяет действия программы на каждом ее шаге. У оператора нет значения, а у выражения –есть. Каждый оператор заканчивается точкой с запятой.

Составной оператор — это последовательность операторов, заключенная в фигурные скобки.

Блок – это составной оператор, включающий определения переменных в теле блока.

Операции присваивания

Операнд = выражение.

i=i+j;

Допускается многократное присваивание слева направо.

i=j=2+(k=3); получим: сначала к=3, потом j=5, потом i=5

Арифметические операции

***** - умножение, / - деление, % - остаток от деления (для целых),

+ - сложение, - вычитание.

Возведение в степень в *C++* отсутствует, но существует специальная функция.

Приведение типов

Если оба числа целого типа, то результат будет – целое. Если переменные разного типа, то результат будет приведен к более общему.

int a=3, b=2;

double z=a/b; // Сначала получим результат деления =1 (целую единицу), а приведение к double выдаст окончательный результат 1, (вещественную единицу).

Операции ++ и – -

Операции приращения для переменной целого типа приводят:

- к увеличению ++ (increment),
- к уменьшению -- (decrement) значений переменной на 1.

Причем если операция стоит перед переменной, то она будет произведена перед использованием переменной, если после – то после ее использования.

i=0;
j=++i; // j=1, i=1
k=i--; //k=1, i=0

Битовые операции

осуществляются над переменными, рассматриваемыми как битовые цепочки, и применяются для переменных целого типа *char, short, int, long*. Битовые операции производятся поразрядно и без переносов.

&- битовое умножение. 10010011
00111101
Результат ***00010001***

|- битовое сложение. 10010011
00111101
Результат ***10111111***

^- *сумма по модулю 2.* 10010011
00111101
Результат ***10101110***
~- *битовое отрицание.* 00111101
Результат ***11000010***
<<- *логический сдвиг влево* (00000001) <<4 =***000100000***
>>- *логический сдвиг вправо* (00010000) >>4 =***000000001***

Примеры использования битовых операций
i<<1 // умножить на 2 (i=0011=3_{10}..>>>0110=6_{10})
i>>2 // поделить на4 (i=1100=12_{10}..>>>0011=3_{10})
i & 1 //проверка на четность (i=0011=3_{10}..>>>0010=2_{10})
i | 128 // уст-ть 1 в 7-ой бит (i=0000000=0_{10}..>>>128_{10}=10000000)
(ИЛИ)
I ^ 128 // инвертировать 7-ой бит.

Комбинированные операции

i + = j	//i=i+j	*i & = j*	//i=i&j
i - = j	//i=i-j	*i | = j*	//i=i | j
*i * = j*	//i=i*j	*i << = j*	//i=i<<j
i / = j	//i=i/j	*i >> = j*	//i=i>>j

Операции отношения

Для сравнения значений переменных используются операции отношения.

Результатом будет True или False.

>	больше
<	меньше
>=	больше или равно
<=	меньше или равно
==	равно
!=	не равно

Логические операции

Применяются над логическими операндами, в качестве которых могут выступать операции отношения.

&&	И, истинно, если аргументы и слева и справа являются истиной.
||	ИЛИ, истинно, если аргументы или слева, или справа являются истиной.
!	НЕ, истинно, если аргумент принимает ложное значение.

3.6. ОПЕРАЦИИ В С

3.6.1. Условный оператор if…else

Оператор используется для разветвления процесса вычислений на два направления. Существует две формы записи данного оператора, которые представлены схемами на рис.3а) - *сокращенная форма* и рис.3б) – *полная форма*.

«Операторы 1, 2» могут быть как *простыми*, так и *составными*. *Составной* оператор записывается в фигурных скобках

{*составной оператор 1****;}***

и включает несколько операторов, которые должны быть выполнены при выборе соответствующей ветви программы.

Если необходимо выполнить только один оператор (т.е. при работе с простым оператором), то его можно не заключать в фигурные скобки.

Пример использования полной формы записи оператора *if-then* и результата работы программы для получения модуля числа представлен на рис.3.23.

Полная форма записи	***Сокращенная форма записи***
*if (*логическое выражение*) {*оператор 1*;}* *else {*оператор 2*;}*	*if* ***(***логическое выражение***) {***оператор 1***;};**

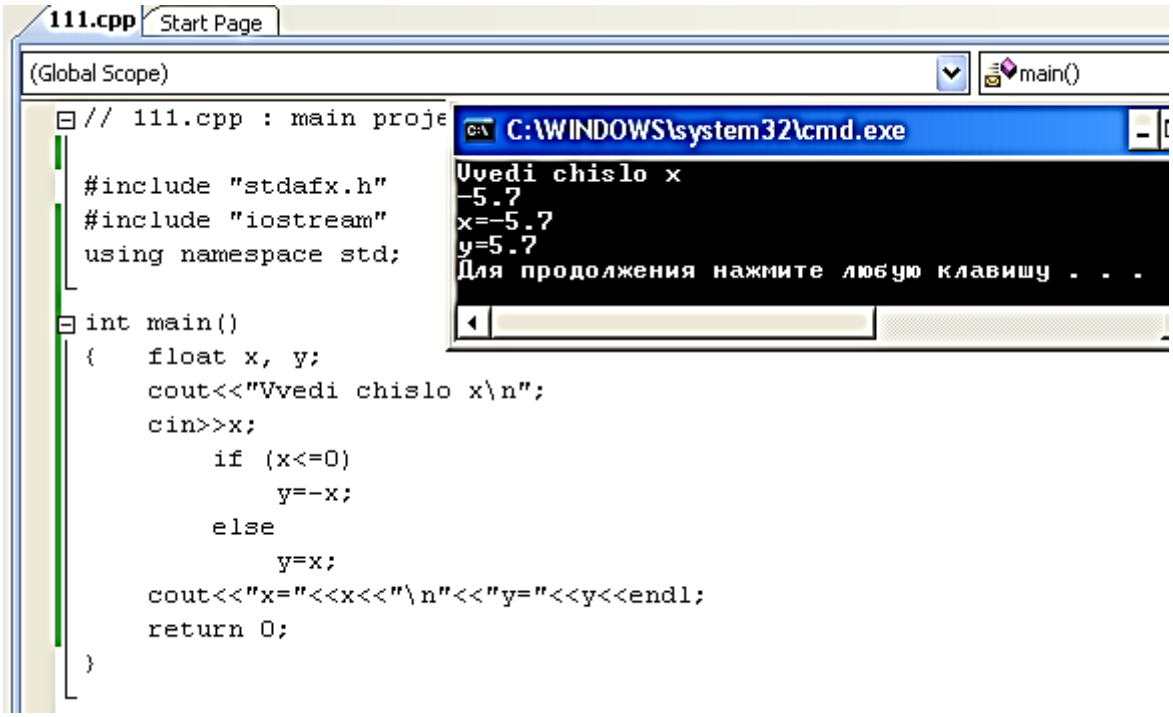


Рис.3.23 - Листинг программы с оператором *if-then*

В некоторых случаях условный оператор удобнее использовать, ориентируясь на формат:

(условие) ? выражение 1: выражение 2 ;

Пример использования условного арифметического оператора:

если *i* положительно, то переменной *a* следует присвоить значение *100*, если отрицательно, то *200*.

```
If (i>=0)   a=100; else a =200;         // обычная форма
a = (a>=0) ? 100: a =200;       // условный арифметический оператор.
```

3.6.2. Оператор-переключатель

Задача – *выбор одного варианта из многих* может быть решена с помощью нескольких вложенных друг в друга операторов *if….else*, но более удобный способ – использование *оператора-переключателя switch*, синтаксис которого выглядит так:

```
switch(«выражение»)
   {
     case «константа 1»: «оператор 1»;   [break;]  //ветвь 1
     case «константа 2»: «оператор 2» ;  [break;]  //ветвь 2
      ……
     case «константа n»: «оператор n»;  [break;]
     [default : «оператор n+1»;]
   }
«оператор k»;
```

Этот оператор выполняется так.

- Сначала вычисляется значение *«выражения»*, тип которого должен быть одним из простых (*int, char).*
- Вычисленное значение (полученное число или символ) сравнивается со значением константного выражения *«константа 1»*. При совпадении этих двух значений выполняется соответствующий *«оператор 1»*, который может быть составным, и управление передается оператору, следующему сразу после *switch* – «*оператор k*». Это происходит в том случае, если в соответствующей ветви имеется необязательный оператор *break.*
- Если такого оператора в ветви нет, то управление передается последовательно на следующую ветвь *(ветвь 2)*. Здесь при совпадении «*выражения*» и *«константы 2»* будет выполнен *«оператор 2»*. Если совпадения не произошло, то управление передается на следующую ветвь. Так будет продолжаться до тех пор, пока не будет выполнен «*оператор n+1*».
- Если значение «*выражения*» не совпало ни с одним из значений «*констант*», то выполняется ветвь, помеченная оператором *default*. При ее отсутствии выполняется следующий за *switch* оператор -«*оператор k*».

Пример использования оператора приведен на рис.3.24

Задание 3.7

3.7.1 Напишите программу для расчета среднего арифметического трех целых чисел, вводимых с клавиатуры.

3.7.2 Напишите программу для расчета среднего арифметического трех вещественных чисел, вводимых с клавиатуры.

```
// progr1_2008.cpp : main project file.

#include "stdafx.h"
#include "iostream"
using namespace std;

int main()
{
    int a;
    cout<<"Введите целое число от 1 до 3\n";
    cin>>a;
    switch (a)
    {
    case 1: cout<<"ввели 1\n";break;
    case 2: cout<<"ввели 2\n";break;
    case 3: cout<<"ввели 3\n";break;
    default: cout<<"Неверный ввод";
    }
}
```

Рис.3.24 - Листинг программы с оператором - переключателем *Switch*

3.7.3 Напишите программу для расчета площади круга. Исходные данные вводить с клавиатуры.

3.7.4 Напишите программу для расчета площади прямоугольника. Исходные данные вводить с клавиатуры

3.7.5 Напишите программу для расчета площади треугольника. Исходные данные вводить с клавиатуры.

а) $S=b*h_b/2$, где b- основание, h_b -высота проведенная к основанию.

б) $S=(a*b*\sin\alpha)/2$, где a и b- стороны треугольника, α- угол между ними.

в) $S=a*b*c/R$, где a, b, c- стороны треугольника, R - радиус описанной окружности.

г) $S=r*p$, где $p=(a+b+c)/2$, где r -радиус вписанной окружности.

д) $S=(p(p-a)(p-b)(p-c))^{0,5}$, где $p=(a+b+c)/2$.

3.7.6 Напишите программу для расчетов параметров фигур. Исходные данные вводить с клавиатуры.

а) Центральный угол сектора. $\alpha=2*(2hr-h^2)^{0,5}$, где h- стрела сегмента, r - радиус сегмента.

б) Стрела сегмента $h=r-(r^2-(a^2/4))^{0,5}$, где a- длина хорды

3.7.7 На рисунках ниже на плоскости представлены прямые, окружности и парабола.

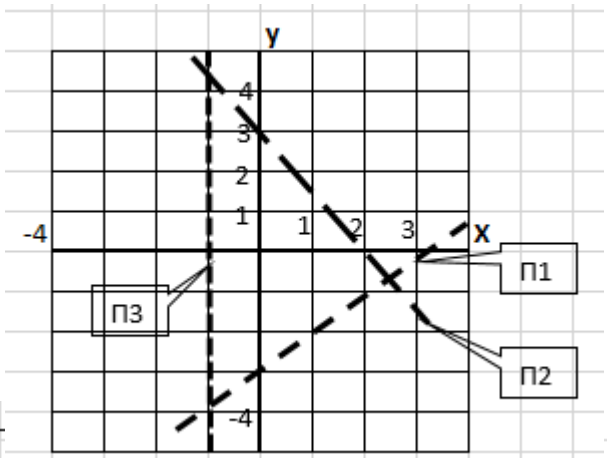
Ваша задача – написать программу, в которой:

- вводятся 2 вещественных числа (координаты: x, y);

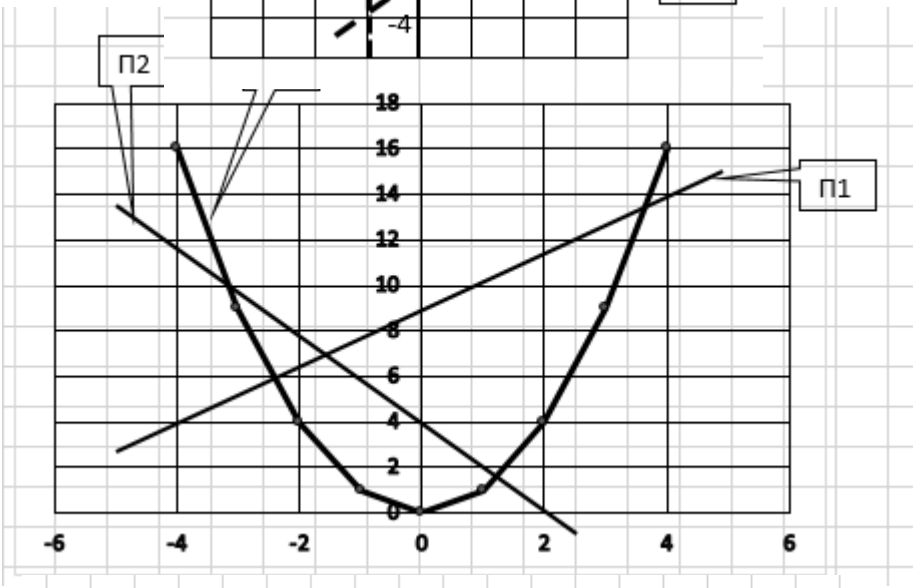
- определяется попала ли точка с заданными координатами в область, ограниченную заданными линиями;

- рисунок и линии определяет преподаватель.

а)



б)



в)

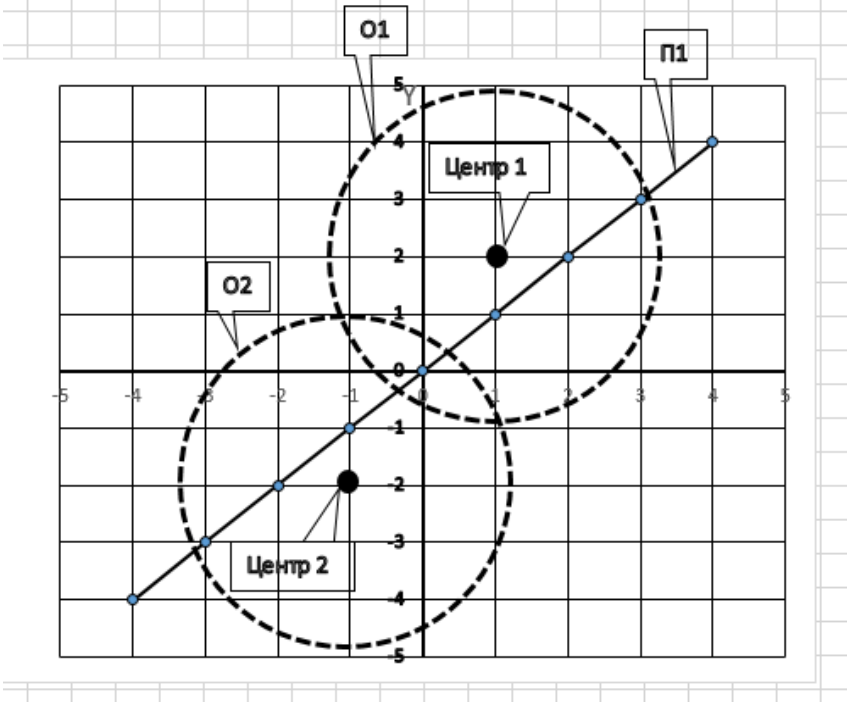


Рис.3.25 - Рисунок для задания 3.7

3.7.8. Напишите программу-калькулятор для 4-х арифметических действий (+, -, *, /), используя оператор:

- if…else,
- switch.

Аргументы (целые числа) и арифметическое действие задавать с клавиатуры.

3.7.9. Напишите программу, которая производила бы битовое сложение двух чисел, заданных с клавиатуры.

3.7.10. Напишите программу, которая производила бы битовое умножение двух чисел, заданных с клавиатуры.

3.7.11. Напишите программу, в которой проверялась бы операция отношения. Числа для вариантов вводить с клавиатуры.

а) операнды - целые числа,

б) операнды - логические переменные.

3.6.3. Операторы цикла

Почти любая программа не обходится без использования алгоритмов циклических структур.

Циклы бывают (см. п.п. 1.3):

- *с предусловием* *(while* //оператор цикла с предусловием),
- *с постусловием* *(do…while,* //оператор цикла с постусловием),
- *обобщенный* (параметрический) цикл *(for…)*

Рассмотрим эти операторы более детально.

3.6.3.1. Оператор цикла “while”

Структурная схема, описывающая работу оператора представлена на рис.2.4 а). Синтаксис оператора реализации цикла с предусловием имеет вид:

while (условие) оператор;	для простого оператора;
while (условие) {оператор;}	для составного оператора.

Выполнение оператора трактуется так:

«*до тех пор, пока «условие» имеет истинное значение, выполнять «оператор» – тело цикла,*

иначе (условие приняло ложное значение) – выйти из цикла.

Отличительной особенностью циклического алгоритма с предусловием является то, что в этом цикле выполнение операторов, входящих в его тело, может не произойти вообще, если при первой же проверке условие выполнения тела цикла окажется недопустимым.

Задание 3.8

Введем 3 числа и найдем их среднее арифметическое.

Текст программы для этого случая представлен на рис. 3.26. На нем все операторы прокомментированы.

Ваша задача сводится к тому, чтобы набрать текст программы в редакторе *C++*, отладить программу и запустить на выполнение.

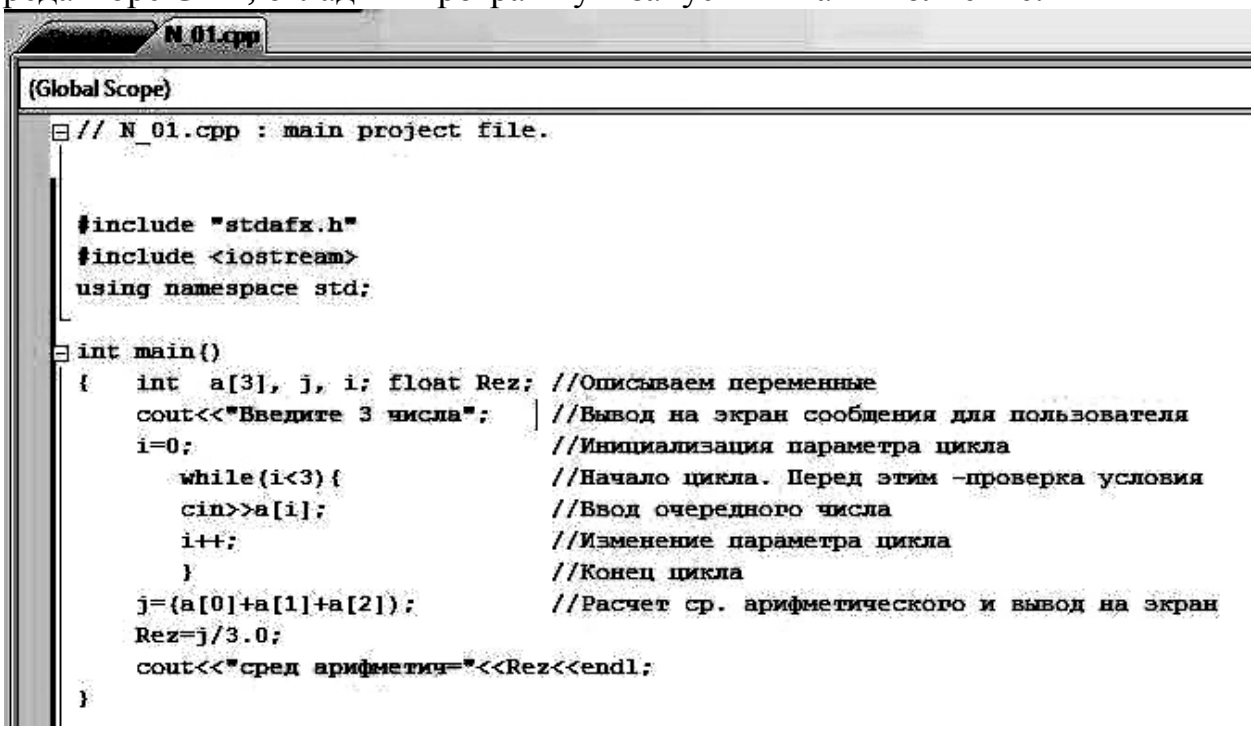


Рис.3.26–Листинг программы для цикла с предусловием

3.6.3.2 Оператор цикла “do…while”

Структурная схема, описывающая работу оператора, представлена на рис.2.4 б)

Синтаксис оператора реализации цикла с постусловием:

do {операторы;}while (условие); для составного оператора,

do оператор while выражение; для простого оператора,

Смысл трактовки этого оператора в следующем:

выполняется (do) простой или составной оператор до тех пор, пока (while) логическое условие имеет истинное значение (выражение принимает значение – истина).

Отличительной особенностью циклического алгоритма с постусловием является то, что в этом цикле выполнение операторов, входящих в его тело, произойдет хотя бы один раз. Такой цикл может быть с заданным числом повторений и с неопределенным числом повторений.

Согласно структурной схеме и синтаксису словесный алгоритм цикла с постусловием и с заданным числом повторений трактуется и записывается в такой последовательности:

1) Задать начальное значение параметру цикла (его можно назвать переменной-счетчиком итераций). Этот параметр является переменной либо целого типа, либо символьного типа;

2) Записать ключевое слово *do,* после которого открыть фигурную скобку «***{***». Фигурная скобка открывает тело цикла. Если тело состоит из нескольких операторов, то по правилам синтаксиса операторов тело цикла – это составной оператор;

3) Последовательно записать операторы, составляющие тело цикла;

4) Последним оператором тела цикла должен быть оператор, изменяющий параметр цикла - *счетчик*. После него закрывается фигурная скобка «***}***» – конец тела цикла;

5) Осуществить проверку логического условия – сравнить достиг ли параметр цикла требуемого значения. Если результат проверки истинный (т.е. параметр цикла не достиг искомого значения), то цикл повторяется. Если результат проверки ложный, т.е. параметр цикла достиг искомого значения, то происходит выход из цикла.

Задание 3.9

Как и предыдущем случае рассчитаем среднее арифметическое 3-х чисел.

Параметром цикла является счетчик итераций i..

Программа для этого случая будет отличаться от предыдущей только операторами, характерными для рассматриваемого цикла. Они приводятся ниже.

```
i= 0;          //Установка параметра цикла (ПЦ)-(начальные установки)
do{.           /* Начало цикла (фигурная скобка открывает составной
                оператор, являясь при этом началом тела цикла) */
cin>>a[i];
i++;           //Изменение параметра цикла (ИПЦ)
}              //Конец составного оператора  (тела цикла)
while(i<3);  //Проверка условия и  конец цикла
```

Напишите программу для определения среднего арифметического, используя данный оператор цикла. Добейтесь ее работоспособности.

3.6.3.3. Оператор цикла “for”

Структурная схема, описывающая работу оператора, представлена на рис.2.5.

Синтаксис оператора ***for*** имеет вид, представленный ниже (это 2 варианта записи синтаксиса, хотя они одинаковы):

*for**(**«выражение 1»**;** «условие»**;** «выражение 2»**)** **{**«оператор 1»**;}***

*for**(** инициализация**;** выражение или проверка условия**;** модификация**)** оператор 1**;***

«Выражение 1» определяет действия, выполняемые до начала цикла, т.е. определяет *начальные условия для цикла –инициализацию***.** Вычисляется только один раз.

Логическое выражение *«условие*» определяет условия окончания или продолжения цикла: цикл продолжается, если значение этого логического выражения *истинно*.

«*Выражение 2» или модификация* обычно задает необходимые для следующей итерации *изменения параметров цикла***.**

Задание 3.10

Повторите программу расчета среднеарифметического 3-х введенных чисел, используя оператор *for*.

Команды программы, отличающие его от двух рассмотренных, приведены ниже.

```
for(i=0;i<3;i++)  /* Для i от 0 (это инициализация),
                     до тех пор, пока i будет оставаться менее 3 (это
                     условие) прибавлять к i единицу (модификация) */
cin>>a[i];        // Операторы
```

Нетрудно видеть, что во всех трех случаях результат работы программы одинаков.

Задание 3.11

Используя операторы цикла (выбор оператора производит преподаватель) выполнить следующее:

1. Ввести с клавиатуры массив из 7 элементов.

   а). Выбрать из массива элементы, значения которых менее 3 и распечатать их на экране.

   б). Введенными значениями заполнить второй массив. Первый элемент введенного массива должен стать последним элементом второго, второй элемент веденного – предпоследним элементом второго и т.д.

   в). Сформировать второй массив из первого так, чтобы во второй массив попали только отрицательные числа.

   г). Найти во введённом массиве минимальное число.

   д) Найти во введенном массиве максимальное число.

   е) Расположить числа введенного массива по возрастанию.

2). Написать программу, которая должна вводить с клавиатуры слово из 3-х букв. Это слово должно сравниваться в программе с задуманным словом. При совпадении этих двух слов необходимо вывести на экран соответствующую информацию. Количество попыток ввода слова для угадывания должно быть три. Если после трех попыток совпадение слов не произошло, программу закончить.

3.7. ВВОД И ВЫВОД В С++

3.7.1. Общие сведения о системе ввода/вывода С++

Система ввода/вывода С++ действует через *потоки (streams).* *Поток ввода/вывода* – это логическое устройство, которое выдает и принимает пользовательскую информацию.

Поток связан с физическим устройством с помощью системы ввода/вывода *С++*. Поскольку все потоки ввода/вывода действуют одинаково, то, несмотря на то, что программисту приходится работать с совершенно разными по характеристикам устройствами, система ввода/вывода предоставляет для этого единый удобный интерфейс.

Например, функция, которая используется для вывода информации на монитор, вполне подойдет для вывода на принтер и для записи в файл.

Когда начинает выполняться программа на *С++,* автоматически открываются *три потока*:

cin – стандартный ввод (с клавиатуры);

cout – стандартный вывод (на экран);

cerr – стандартная ошибка (на экран).

Потоки *cin* и *cout* нами использовались в программах как стандартные средства обмена информацией с клавиатурой и монитором (с консолью). Этот стандартный ввод/вывод называется консольным. Чтобы обеспечить данный ввод/вывод, к программе подключается заголовочный файл *<iostream.h>*.

Потоками можно управлять потоками при помощи *манипуляторов* (инструкций форматирования), которые вставляются прямо в поток. О таком форматировании информации можно более подробно узнать из книги Литвиненко Н.А. «Технология программирования на С++. Начальный курс» [15].

Для использования манипуляторов необходимо добавить файл включений *–iomanip.h*

Полный список манипуляторов представлен на стр. 117 [15]. Некоторые приведены ниже.

- *setw(n)* - манипулятор для установки ширины поля вывода в n символов.

- *dec* – перевод в десятичную систему,

- *oct* - перевод в восьмеричную систему,

- *hex* - перевод в шестнадцатеричную систему,

- *setfill(char)* – установить символ - заменитель,

- *setpresision(n)* – количество цифр после десятичной точки.

До сих пор нами в программах для вывода информации использовалась одна (заранее установленная в *C++*) форма (один формат). Однако *C++* позволяет выводить информацию в формах различных вариантов. Организация вывода информации в заранее установленной форме и изменение параметров ввода информации называется форматированием ввода/вывода.

Задание 3.12

Повторите пример использования манипуляторов, приведенный на рис.3.27. Объясните полученные результаты.

```
int main() {

 int a=32767;
        cout<<"a=" <<a<<endl;
        cout<<setw(8)<<dec<<a<<endl;
        cout<<setw(16)<<dec<<a<<endl;
        cout<<setw(8)<<oct<<a<<endl;
        cout<<setw(4)<<hex<<a<<endl;

 double c=2., b=3.;
        cout<<"2/3="<<c/b<<endl;
        cout<<setprecision(4);
        cout<<"2/3="<<setfill('*')<<setw(16)<<c/b<<endl;

return 0;
}
```

```
(Inactive C:\BORLANDC\BIN\QQ_002.EXE)
a=32767
   32767
            32767
   77777
7fff
2/3=0.666667
2/3=**********0.6667
```

Рис.3.27 - Использование манипуляторов в программе

3.7.2. Форматный ввод/вывод данных

Чтобы использовать функции форматного ввода-вывода, следует воспользоваться в самом начале программы заданием директивы препроцессора *#include<stdio.h>*.

По этой директиве, на этапе компиляции программы, компилятор воспользуется описанием функций, хранящихся в заголовочном файле библиотеки стандартного ввода-вывода *stdio.h* **(***standard input-output head***).**

Рассмотрим основные правила работы с функциями этой библиотеки:
printf( ) - вывода данных на экран,
scanf( ). - ввода данных.

Обратите внимание, что, как и в математике, понятие «функция» предполагает ее имя, последующие за именем скобки, внутри которых могут располагаться аргументы. Аргументов может быть несколько или не быть вообще. О том, что это функции именно форматного ввода-вывода, свидетельствует наличие буквы *f (format)* в конце имени функции.

3.7.2.1. Функция printf( )

Вывод (печать) данных на экран осуществляется в соответствии с синтаксисом функции *printf( ):*

printf("форматная строка", список аргументов);

Форматная строка ограничивается кавычками и может включать в себя:
- *произвольный текст*,
- *управляющие символы* и *спецификации* преобразования данных.

Пример

А) инструкция вида
printf("Ура! Моя первая программа");
– приведет к выводу на экран только текста

Ура! Моя первая программа.

Здесь в качестве параметров функции использовалась только текстовая строка.

Б) Если же выполнить функцию
printf("Ура! \nМоя первая программа");
– то результатом ее работы будет вывод текста на экран в следующем виде:

Ура!
Моя первая программа

Такое (построковое) представление произошло потому, что в форматной строке был задан управляющий форматом вывода символ \n, переводящий курсор (точку вывода) на следующую строку экрана.

Кроме этого управляющего символа чаше всего используются также следующие символы:

\t – горизонтальная табуляция;
\r – возврат каретки(курсора) к началу строки (без перевода каретки на новую строку);
\\ – обратная косая черта \, применяемая обычно для задания пути к файлам;
\' – апостроф (символ одиночной кавычки ‘);

\" – кавычка (символ двойной кавычки ***").***

Использование указанных символов приводит к достаточно эффективной организации вывода данных. Например:

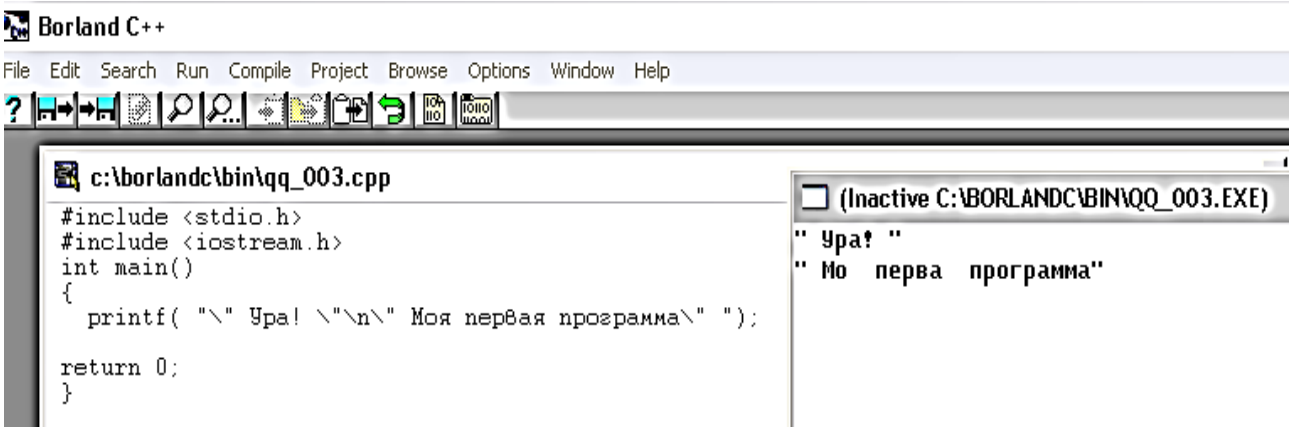


Рис.3.29 - Форматный вывод

Для более понятного представления функций *форматного ввода/вывода* на рисунке рис.3.30 приведем несколько примеров их сравнений с операторами *потокового ввода/вывода*. Выражение и результаты действия этих конструкций эквивалентны.

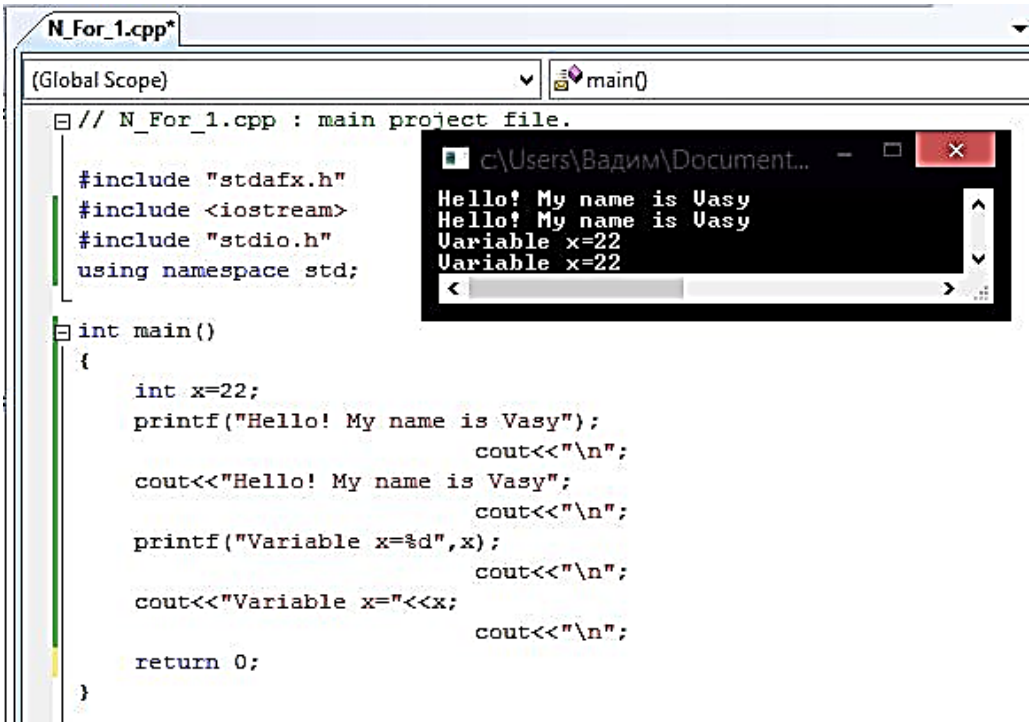


Рис.3.30 - Форматный и потоковый вывод информации

Рассмотрим подробнее форматный ввод (вывод на экран). Его представляет инструкция:

Форматная строка предназначена -для того, чтобы определить:

- ✓ какого типа значение должно быть выведено,
- ✓ какой переменной это значение должно принадлежать,
- ✓ как выведенное значение переменной должно выглядеть на экране.

Обобщенный формат спецификации преобразования имеет вид:
% ширина_поля.точность спецификатор,
причем обязательными являются только два:
% и спецификатор.

Для нашего случая в форматной строке
%d – это спецификация преобразования данных.

Спецификаторы форматной строки для функции форматного вывода:

d – для целых десятичных чисел со знаком (см пример выше);

u – для целых десятичных чисел без знака (unsigned);

f – для вещественных чисел в формате с фиксированной точкой; *(знак_dddd.dddd, количество цифр перед точкой зависит от величины числа, после точки - от точности преставления.)*

e – для вещественных чисел в форме с плавающей точкой (*знак_m.dddd**e**знак_xxx, m - одна десятичная цифра, dddd-цифры после точки, **e** - признак порядка, xxx-числа для указания порядка);*

s – символьная строка. Символы печатаются либо до первого нулевого символа ‘\0’, либо печатается то кол-во символов, которое задано в поле точность спецификации.

Задание 10

Рассмотрите пример использования спецификаторов (рис.3.31):

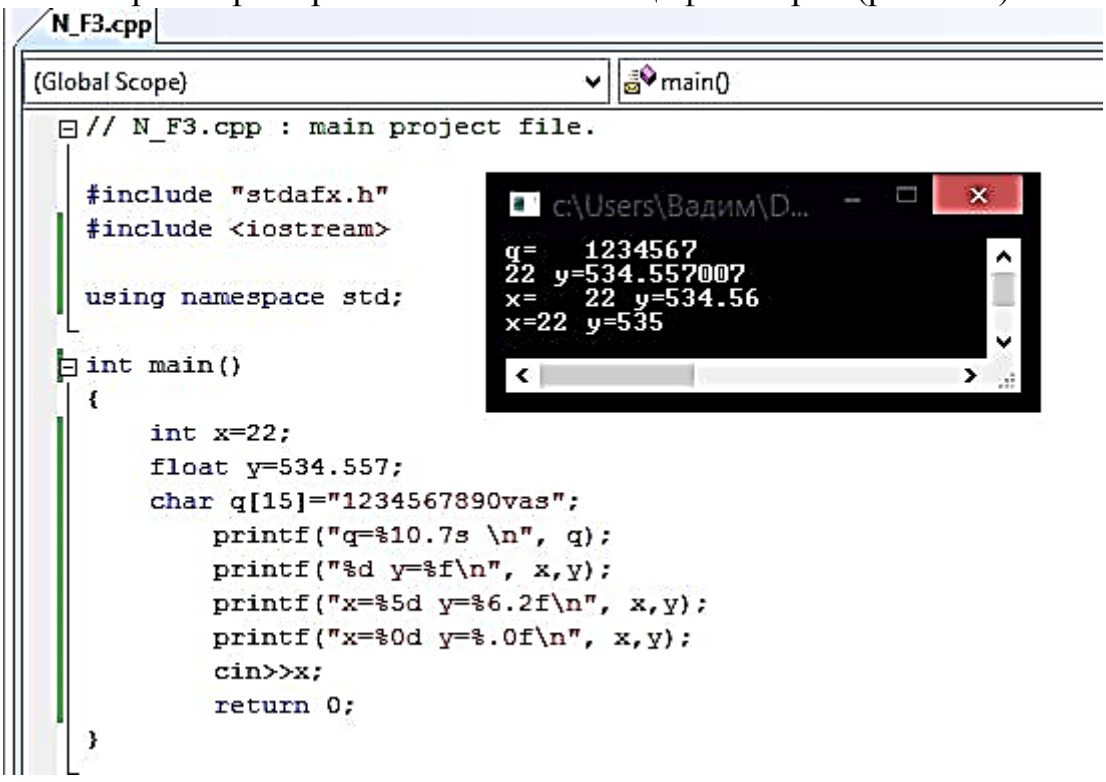


Рис.3.31 - Использование спецификаторов при вводе-выводе

Замечания.

При выводе: в первой строке ширина поля *10* символов, а точность -*7*;
во второй строке отсутствует *“x=”* ;
в третьей строке- под *x* отвели *5* знаков;
под *c*- 6 (это ширина поля вывода), причем два знака после десятичной точки определяют *точность* для вещественных чисел;
в четвертой строке под дробную часть вообще ничего не отвели.

Повторите приведенный пример. Выполните задание, которое получите у преподавателя.

3.7.2.2. Функция scanf( )

Функция применяется для *ввода* данных с клавиатуры.

Ее синтаксис имеет вид:

scanf(“форматная строка”, список аргументов);

Здесь форматная строка имеет вид почти аналогичный виду спецификации вывода, т.е.:

% ширина_поля спецификатор

при этом обязательными являются % и спецификатор.

Аргументами для функции ввода могут быть только *адреса объектов программы,* в частном случае это *адреса переменных***.**

Для обозначения адреса переменной перед ее именем указывается знак &.

Таким образом, чтобы, например, ввести с клавиатуры три числа разного типа (пусть a=10 – целое, b=22 - вещественное, c=33.3 - вещественное) следует воспользоваться такой нотацией функции ввода:

scanf(**“**%d%f%f”**,** &a,&b,&c**);**

В результате по адресам памяти, отведенным под хранение значений соответствующих переменных, будут занесены соответственно значения *a=10, b=22.0, c=33.3.*

Пример эквивалентных записей для ввода данных:

scanf(“%d%f%f”, &a,&b,&c); *и* *cin >> a >> b >> c;*

3.7.3. Файловый ввод/вывод

3.7.3.1 Основные понятия файлового ввода/вывода

Под файлом понимается именованная область памяти (обычно на диске: FDD, HDD, CD), которая рассматривается в компьютере как единое целое. *Для компилятора файл – это сложный тип данных.*

В файл можно записывать данные – *вывод в файл*; из файла можно считывать данные – *ввод из файла*.

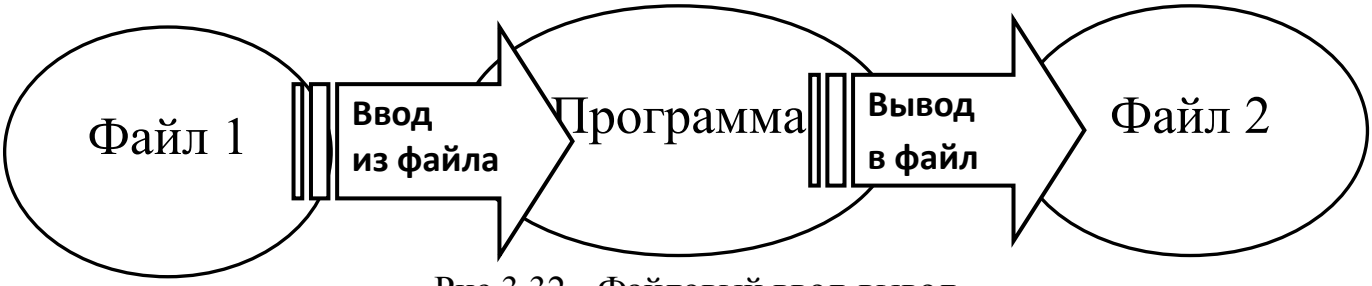


Рис.3.32 - Файловый ввод-вывод

Задача формулируется следующим образом: необходимо сделать так, чтобы файлы с именами, которые придумал пользователь (к ПК они в принципе отношения не имеют), можно было читать инструкциями чтения данных, зашитыми в памяти машины (в ее библиотеках).

Для этого сначала открывают файл с помощью библиотечной функции *fopen, (из биб-ки <stdio.h>.)* Эта функция получает внешнее имя файла и возвращает *указатель* (например, **fp)* для доступа к файлу.

Файловый указатель ссылается на структуру, содержащую информацию о файле (адрес буфера, положение текущей литеры в буфере, файл открыт для чтения или записи, встретился ли конец файла).

Следующий вопрос: как читать из файла или писать в файл, если он открыт?

Существует несколько способов:

1. *int getc(FILE *fp)* функция возвращает следующую литеру из файла, на который ссылается указатель **fp*. В случае ошибки вернет *EOF;*
2. *int putc(int c, FILE *fp)* функция пишет литеру ***c*** в файл *fp*. В случае ошибки или исчерпания файла функция вернет *EOF;*
3. *int fscanf(FILE *fp, char *format,…)* функция форматного ввода (получение данных из файла);
4. *int fprintf(FILE *fp, char *format,…)* функции форматного вывода (запись данных в файл), Аналоги *scanf и printf,* но с той разницей, что ссылаются на файл.

3.7.3.2. Порядок и правила выполнения файлового ввода и вывода

*1 этап. Определение логического файла***.**

Это делается с помощью такого оператора описания:

*FILE *<переменная>.*

Например, *FILE *fp;*

2 этап. Устанавливается связь между объявленным *логическим* файлом и файлом *физическим.*

Это делается с помощью библиотечной функции открытия файла *fopen().*

Ее синтаксис имеет вид:

fopen(<"имя физического файла">,<"режим использования">),

где под *именем физического файла* понимается не только непосредственно имя, но и полный путь к тому месту, где этот файл хранится.

Например, *fopen("C:\\Windows\\Temp\\file1.txt",<"режим использования">)*

означает, что открываемый файл с именем *file1.txt* хранится на диске *C*, в каталоге *Windows*, в подкаталоге *Temp*.

Обратите внимание, что в указании пути вместо одного слеша используется два слеша **«\\».**

Режим использования физического файла – это параметр, определяющий, как будут обрабатываться файловые данные. Этот режим может принимать следующие *значения (*наиболее часто употребляемые):

- ✓ *"a"* – файл открывается для записи в конец файла (добавление), если физически файл не существует, то он сначала создается;
- ✓ *"r"* – файл открывается только для чтения из него данных. Физический файл при этом должен существовать, иначе работа программы завершится аварийно по признаку отсутсвия файла;
- ✓ *"w"* – файл открывается только для записи. При этом, если файл отсутствует, то он создается. Если такой файл уже создан, то он перезаписывается без сохранения в нем старых данных, так как запись осуществляется с начала физического файла.

Таким образом, например, такая запись внутри тела функции

fp=fopen("C:\\Windows\\Temp\\file1.txt","r")

означает, что открываемый файл с именем *file1.txt* хранится на диске *C*, в каталоге *Windows*, в подкаталоге *Temp*. Файл открывается для чтения из него данных, а в программе на языке Си работа с этими данными будет происходить через логический файл *fp*. Таким образом, *мы связали физический file1.txt и логически fp файлы.*

3 этап. Работа с данными.

Если файл открывается в режиме чтения, то осуществляется проверка на его существование.

На этом этапе записываем данные в файл или читаем из него.

4. этап. Освобождение буфера файла при помощи функции *fclose(<имя логического файла>)* после завершения обмена с данными, т.е. происходит закрытие файла.

3.7.3.3. Файловый вывод – функция fprintf (запись данных в файл (вывод в стандартный выходной поток))

Прототип функции имеет вид:

fprintf(Указатель_на_поток, форматная_строка,
список_переменных);

От рассмотренной выше функции *printf()* для форматного обмена данными с дисплеем эта функция отличается тем, что в качестве первого параметра в ней необходимо задать указатель на поток, с которым производится обмен.

Задание 3.13. Составить программу, позволяющую записать в файл с именем C:\Test\cell.txt два вещественных числа (см.рис.3.33).

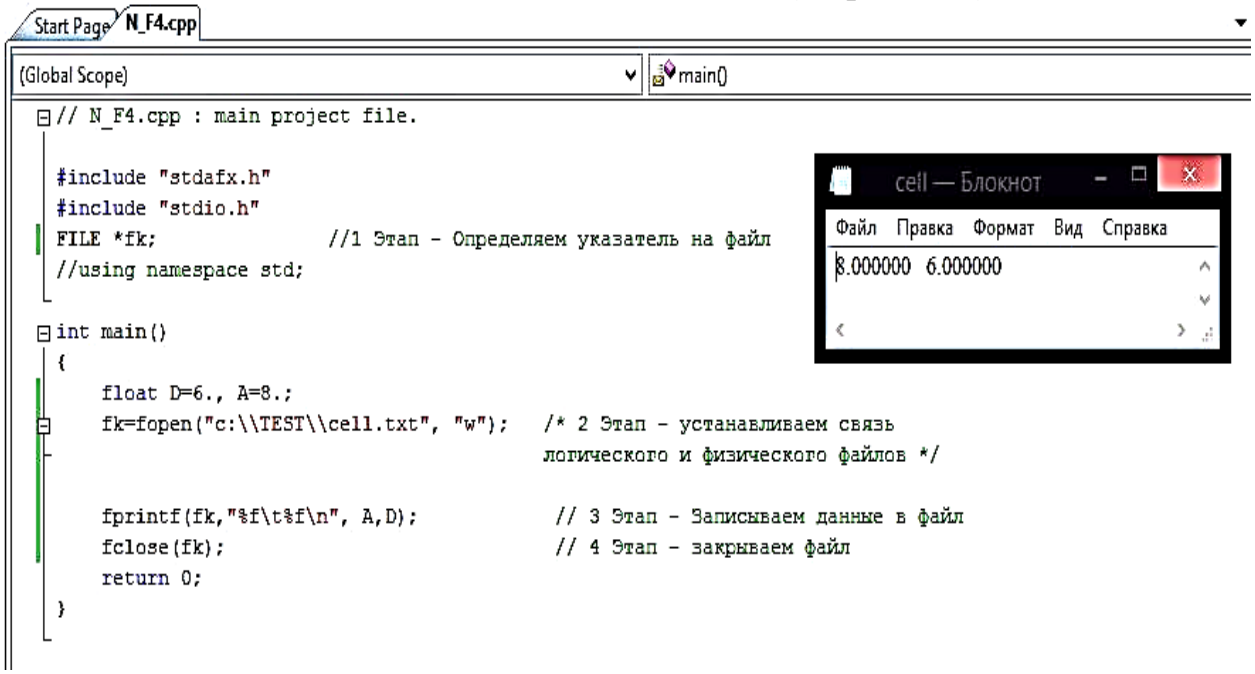


Рис3.33 - Использование функции *fprintf*

Справа в отдельном окне показано содержимое файла, в который была записана информация. Это этого использован «Блокнот».

Здесь функция *fprintf(fk,"%f\t%f\n",A,D);* пересылает данные в открытый в режиме записи файл *fk (это логическое имя файла)* значения переменных *A, D.*

Эти переменные, имеющие вещественный тип, будут размещены последовательно, начиная с первой его позиции, и будут разделены друг от друга признаком табуляции *(\t).*

В этом примере не выполнялась проверка на существование файла т.к. мы его сами создавали. Более того, если файл не существует, он будет создан.

В том случае, когда при открытии используется режим типа чтения *(”r”),* следует делать проверку наличия физического файла, чтобы не произошло аварийного выхода из программы.

3.7.3.4. Файловый ввод – функция fscanf (получение данных из файла)

Прототип функции имеет вид:

*fscanf (**у**казатель_на_поток, форматная_строка,*
список_адресов_переменных);

Здесь также присутствует указатель на поток, но вместо имен переменных указывают их адреса. Для того чтобы отличить, что указано в инструкции (команде), используют следующий прием:

- при указании на переменную пишем ее имя- *Name_A,*
- при указании на адрес переменной перед ее именем ставим имперсанд- *&Name_A.*

Задание 3.14. Составить программу, позволяющую считать и вывести на экран два целых числа, записанных в файле с именем *cell.txt* в папке *c:\Test* (см.рис.3.34).

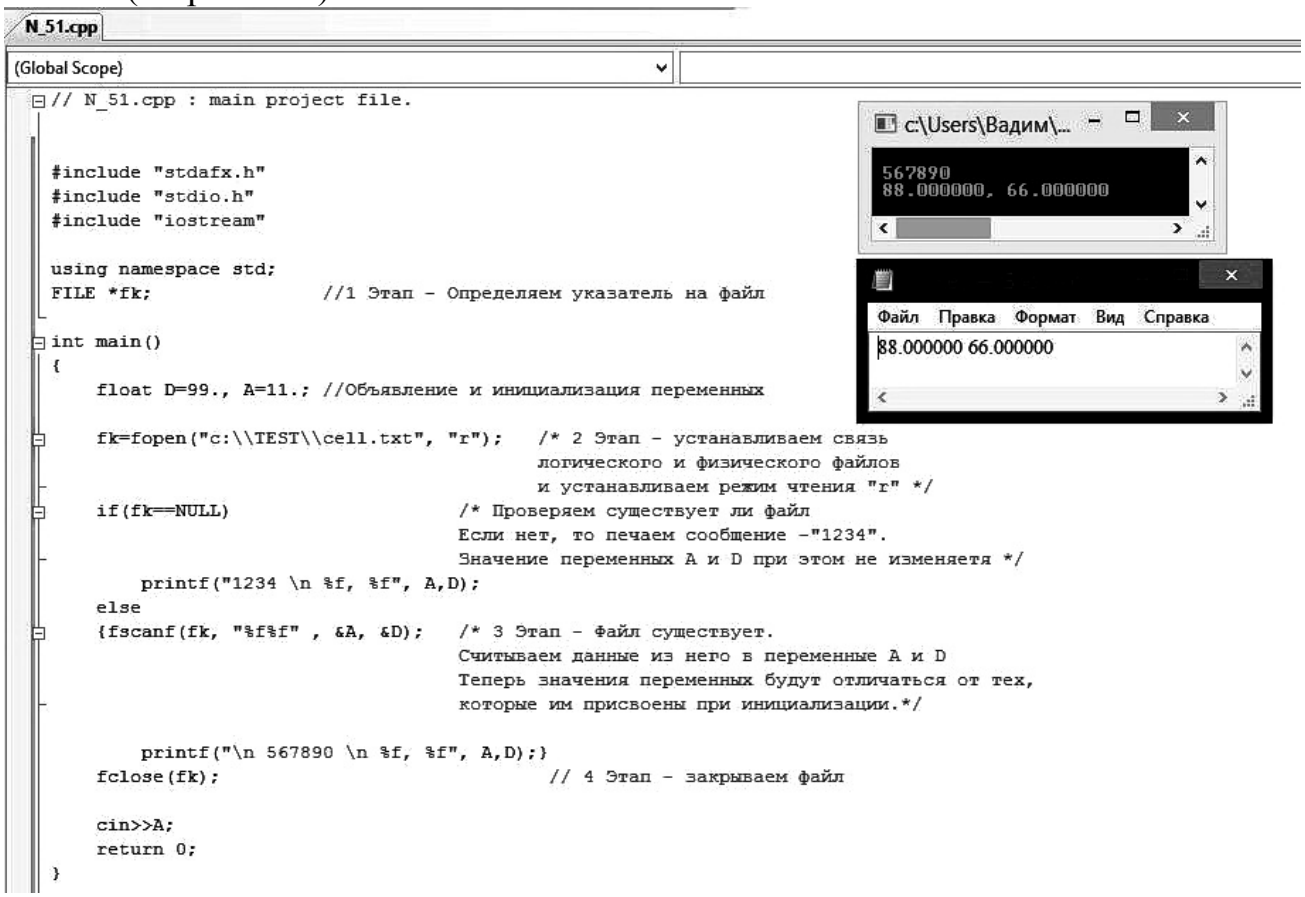


Рис. 3.34 - Листинг программы для случая «файл существует»

Переменным *A* и *D* сначала присваиваем начальные значения. Если файла, к которому обращается программа не существует, то значение переменных не изменится.

Если файл существует, то программа *D* присвоит переменным *A* и новые значения, которые хранятся в файле. Эти значения отличаются от

начальных значений *A* и *D* и вводятся любой программой, например, «Блокнот».

В этом примере константа логического нуля *NULL* означает, что результат работы функции *fopen()* пустой, то есть такой файл не существует. В программе это проверяется по значению, которое было присвоено ранее объявленному логическому файлу *fk.*

Функция *fscanf(fk,"%d%d",&A,&D); вводит данные из* открытого в режиме чтения *файла fk в переменные* A, D, вещественного типа, на что указывает спецификатор *%f.*

Вывод информации возможен в нескольких вариантах.

А). Файла не существует. Поэтому значение переменных A и B, присвоенные им при инициализации, не изменятся.

Б). Файл существует. Формат данных правильный – вещественные числа. Результат: вывод двух целых чисел, хранящихся в файле.

3.7.3.5. Позиционирование в потоке

Кроме функций, которые мы использовали в примерах, часто используются функция:

fseek(<указатель на поток>,<смещение в байтах>,
<позиция указателя или начало отсчета>);

Эта функция осуществляет установку указателя на конкретное местоположение каких-либо данных в файле с помощью задания количества байтов «смещения» от текущей «позиции указателя».

Смещение задается переменной или выражением типа *long* и может быть отрицательным, т.к. возможно перемещение по файлу в прямом и обратном направлении.

Константа типа *long* записывается в виде последовательности десятичных цифр, вслед за которыми идет разделитель ***L*.**

Позиция указателя (начало отсчета) может принимать три целых значения:

- ✓ 0 –указатель установлен на начало файла;
- ✓ 1 – указатель занимает текущее положение;
- ✓ 2 – указатель установлен в конец файла.

Например, записи

fseek(fp,2L,0);
fseek(fp,0L,0);

означают, что в первом случае указатель на местоположение обрабатываемых данных следует установить на позицию, смещенную относительно начала файла на два байта, а во втором случае указатель явно устанавливается на начало файла.

Задание 3.15

1. Напишите и выполните программу, текст которой представлен на рисунке рис.3.35.
2. Инициализируем две вещественных переменных *A и D*. Запишем их в файл.
3. Задав смещение в байтах (1байт -1 символ), запишем в этот же файл те же переменные, но теперь в другом порядке: сначала *D* потом *A*.

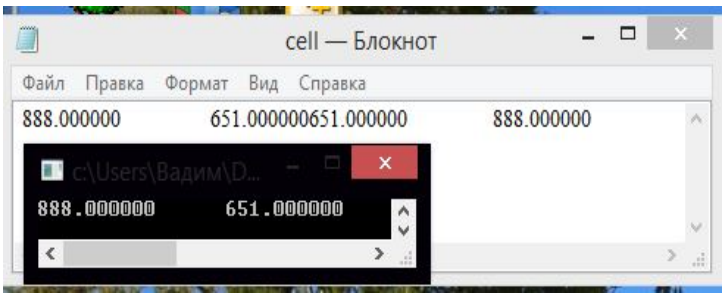
Рис.3.35 -Листинг программы для изучения позиционирования в потоке

Обратите внимание на то, что при нулевом смещении *fseek(fp,0L,0);* запись данных производится на то же место файла, где находились записанные ранее данные. Т.е. происходит стирание старых данных.

Если указать новое место для записи данных в файл, например *fseek(fp,2L,0);*, то в файле два байта (два символа) старых записей будут сохранены и компьютер будет переписывать данные, начиная с третьего байта.

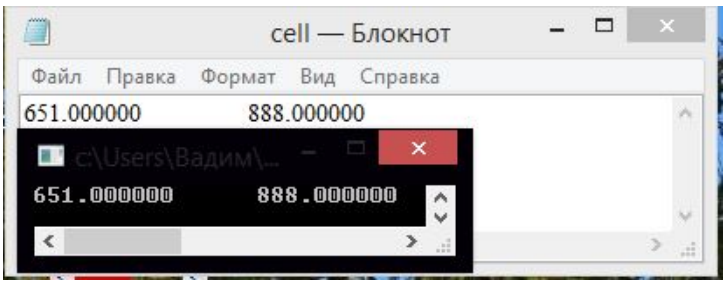
4. Осуществим вывод данных из файла в переменные D1 и A1, причем указав смещение.
5. Выведем полученные значения переменных на экран.
6. Проанализируем содержимое файла cell.txt.
   6.1.Заблокируем операторы 2 и 5.

Новая информация в файл записывается следом за старой без пробелов. При выводе данных из файла считываются первые два числа.

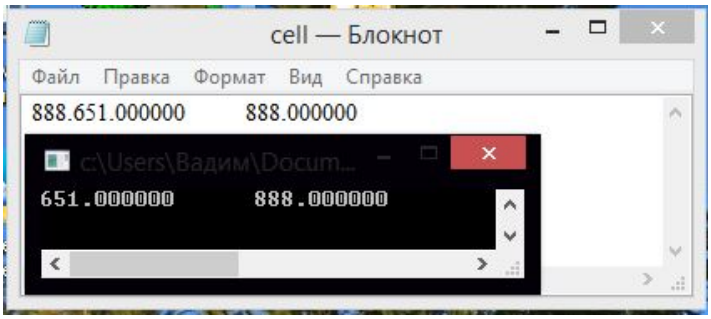


6.2. Разблокируем все операторы.

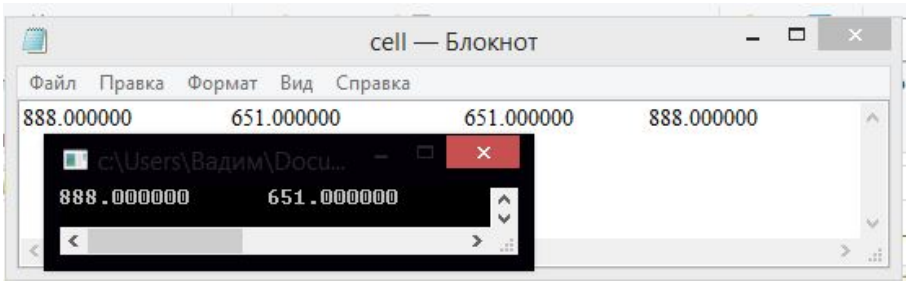
В этом случае запись данных производиться на старое место со стиранием информации.



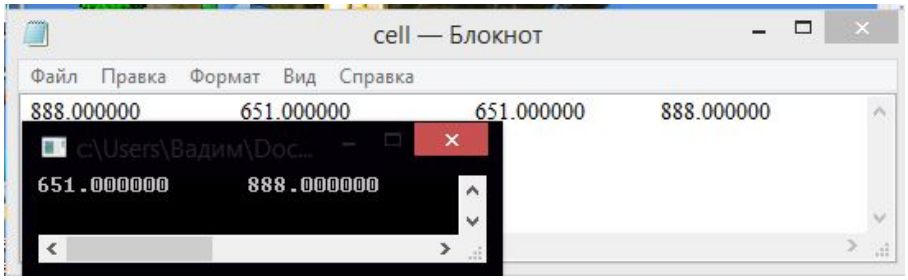
6.3 Изменим операторы (2) и (5) - fseek(fk,4L,0); Сместим точку для записи на 4 байта влево. От старой информации осталось 4 символа.



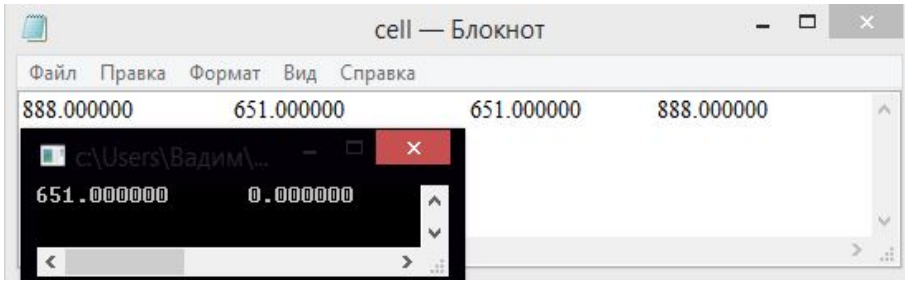
6.4 Изменим оператор (2) и (5) - fseek(fk,40L,0); Сместим точку для записи на 40 байта влево.



6.5. Изменим оператор (2) - fseek(fk,40L,0);) и (5) - fseek(fk,0L,0);



6.6. Изменим оператор (2) fseek(fk,40L,0);) и (5) - fseek(fk,10L,0);



Задание 3.16

1. Составить программу для ввода 6 целых чисел. Вывести их на экран в виде матрицы 3x2 или 2x3. Расстояние между числами должно быть равно интервалу табуляции.

а)- для решения поставленной задачи использовать потоковый ввод-вывод;

б)- для решения поставленной задачи использовать средства форматного ввода-вывода.

2. Составить программу для ввода 6 вещественных чисел. Вывести их на экран в виде матрицы 3x2 или 2x3. Расстояние между числами должно быть равно интервалу табуляции. Количество знаков после запятой -2.

а)- для решения поставленной задачи использовать потоковый ввод-вывод;

б)- для решения поставленной задачи использовать средства форматного ввода-вывода.

3.8. ФУНКЦИИ. СТРУКТУРА ПРОГРАММЫ НА С++

Создавая сложные программы, важно заботиться о том, чтобы с ними было удобно работать. Одним из основных способов обеспечения этого требования является разделение программы на отдельные самостоятельные части. Функция обеспечивает удобный способ отдельно оформить некоторое вычисление и пользоваться им далее, не заботясь о том, как оно реализовано.

Рассмотрим некоторые полезные утверждения и определения.

А). Функция – это логически самостоятельная именованная часть программы, в которую может передаваться любое количество значений переменных аргументов, а возвращаемое функцией значение – только одно.

Б). Вместе с тем, есть функции, в которые не передаются аргументы, и есть функции, которые не возвращают значений.

В). Значение функции – величина переменная и, как всякая переменная, она должна быть объявлена.

Г). Объявление функции производится до ее использования и вне тела любой другой функции. При объявлении функции указывается:
тип_результата имя_функции (список типов параметров);
Естественно, что имя функции задает программист. Желательно, чтобы из имени функции было ясно ее назначение. Такую запись называют *прототипом функции.*

Примеры:

- ✓ *void fun1 (float b, char ch);* - объявлена функция *fun1* с двумя параметрами. Функция не возвращает значения.
- ✓ *float fun2(void);* - функция без параметров, возвращающая вещественное значение.

Д). Объявляются все функции программы, кроме main( ).
Таким образом, если мы создаем в программе функцию, она должна быть объявлена до главной функции *main().* Но это не все. Необходимо указать, что же эта функция будет выполнять, то есть необходимо определить функцию.

Е). Определение функции – это описание операций, которые выполняются в теле этой функции. Оно имеет вид:

Ж). Если функция должна возвращать значение, но *return* отсутствует, то она выдает “мусор”.

З). Функцию можно определять либо непосредственно при объявлении, либо после окончания главной функции *main().* Т.е. структура программы может иметь вид З-А и З-Б, представленный на рис.6.36. Функция *int sq (int a)* возвращает квадрат целого числа.

И). Если определение функции не соответствует объявлению - это ошибка.
Определив функцию, мы указали какие операции эта функция должна выполнять и какие переменные использовать. Осталось указать, в каком именно месте программы эта функция должна использоваться, т.е. где необходимо передать этой функции управление программой. Этот процесс называют *вызовом функции.*

К). Любая программа на *С* начинает выполняться с главной функции *main( ).* Обычно именно из нее вызываются другие функции.

Но в общем случае любая функция может быть вызвана из любой. Одна из функций становится *вызывающей* и временно передает управление *вызываемой* функции, которая, выполнив определенные операции, возвращает управление вызывающей функции.

При вызове функции указывается ее имя и в скобках – список аргументов.

3-А

```
#include<iostream.h>
int sq (int a);          // объявление функции
        void main ( ) //начало описания гл. функции
        {
        ….
        z = sq(x); //Вызов функции
        }
              int sq (int a)  //определение функции
              {
              int b;
              b = a*a;
              return b;       // оператор возврата управления
               }
```

3-Б.

```
#include<iostream.h>
      int sq (int a)          // объявление и определение функции
      {
              int b;
              b = a*a;
              return b;       // оператор возврата управления
      }
void main ( )                 //начало описания гл. функции
{
……..
z = sq(x);                    //Вызов функции
…….. }
```

Рис.3.36 -Объявления и определения функций в программах

«Вызов функции» часто называют *«обращением к функции»;* это равнозначные термины. Обращение к функции следует рассматривать как выражение. Оно может использоваться всюду, где допускаются выражения.

Обратите внимание, что при объявлении и определении функции *square* в качестве аргумент, а мы указали целочисленную переменную *a*. А в программе при вызове функции в нее фактически передали тоже целочисленную переменную, но с именем *x*.

То есть необходимо, чтобы при передаче аргументов обязательно учитывались типы передаваемых и возвращаемых переменных.

Л). Переменную, указанную при объявлении функции, называют *формальным параметром***,** а переменную, передаваемую непосредственно при вызове функции – *фактическим параметром.*
В нашем примере *int a* – формальный параметр, *int x* – фактический параметр.

Задание 3.17. Написать программу для определения квадрата числа $z = x^2$. Возведение числа в квадрат оформить в виде отдельной функции. Доказать ее работоспособность.

3.9. ВНЕШНИЕ ПЕРЕМЕННЫЕ И ОБЛАСТИ ВИДИМОСТИ

Переменные, которые объявлены внутри функции, называются *внутренними переменными или локальными.* Эти переменные могут использоваться только в тех функциях, где они объявлены, и никакие другие функции доступа к ним не имеют.

Каждая локальная переменная функции возникает только в момент обращения к этой функции и исчезает после выхода из нее. Такие переменные называются *автоматическими***.** Они образуются и исчезают одновременно с входом в функцию и выходом из нее; они не сохраняют своих значений от вызова к вызову.

Т.е., обратиться к переменной *b* в теле главной функции нельзя, ее там не «видно». После выхода из функции *square* ячейка оперативной памяти, которая была выделена для этой переменной, освобождается и при следующем вызове под переменную *b* будет лтведена любая другая ячейка.

!!! Т. о., локальные переменные по умолчанию относятся к классу памяти «автоматические» (auto). !!!

Чтобы локальная переменная могла сохранять свои значения от вызова к вызову функции, необходимо переменную отнести к классу памяти *«static»* ***(****статические). Такие переменные не исчезают при выходе из функции.*

Рассмотрим пример. На рис.3.37 представлена программа и результат ее выполнения, который располагается на темных прямоугольниках. Верхний - для статического *k*, нижний - для автоматической переменной.

Статическая переменная *k* инициализируется один раз при первом вызове функции *qqq( ).*

В *''k''* записывается число *2*, которое посредством операции *k+x* увеличивается на *1* и передается в переменную *j*, которая выводится на экран. При втором вызове функции *qqq( )* в нее передается *x=2*, которое добавляется к сохраненному *k=3***.** Функция вернет *5*.

Чтобы переменная не только сохраняла свои значения в течение работы программы, но и ею могли пользоваться другие функции, она должна быть не внутренней, а внешней переменной.

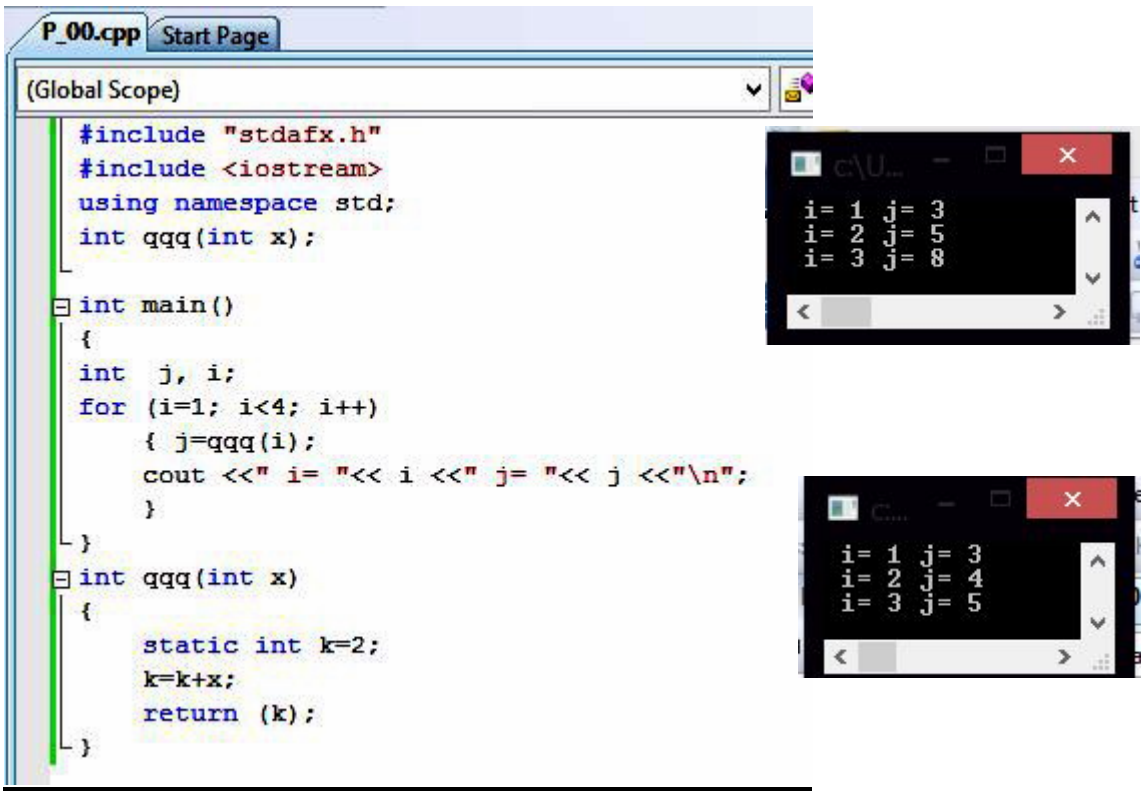


Рис.3.37 - Использование статической переменной

Внешние переменные доступны повсеместно. Их можно использовать в любом месте программы в любой функции. Кроме того, поскольку внешние переменные существуют постоянно, а не возникают и не исчезают на период выполнения функции, свои значения они сохраняют и после возврата из функций, их установивших.

Внешняя переменная должна быть объявлена *вне текста любой функции* и только один раз. В этом случае ей будет выделена память. Обычно внешние переменные объявляются в начале программы после директив препроцессору и перед главной функцией main ( ).

Поскольку внешние переменные доступны всюду, их можно использовать в качестве связующих данных между функциями как альтернативу связей через аргументы и возвращаемыми значениями функций.

Пример. Вычисление квадрата с использованием функции, производящей возведение числа в квадрат. Листинг представлен на рис.3.38.

Связь между функциями осуществляется через внешние переменные. Поэтому в *main( )* присваивается значение переменной *i* и вызывается функция *kvadrat ( )*, которая вычисляет i^2 и присваивает его внешней переменной *j* . Поскольку *main ( )* имеет свободный доступ к переменной *j* ,

можно написать функцию *kvadrat ( )* так, чтобы она не возвращала никакого значения.

Внешние переменные в определенных случаях удобны и эффективны, но пользоваться ими необходимо в меру. Многократное, бездумное использование внешних переменных ухудшает структуру программы и приводит к слишком большому числу связей между функциями, которые не всегда предсказуемы.

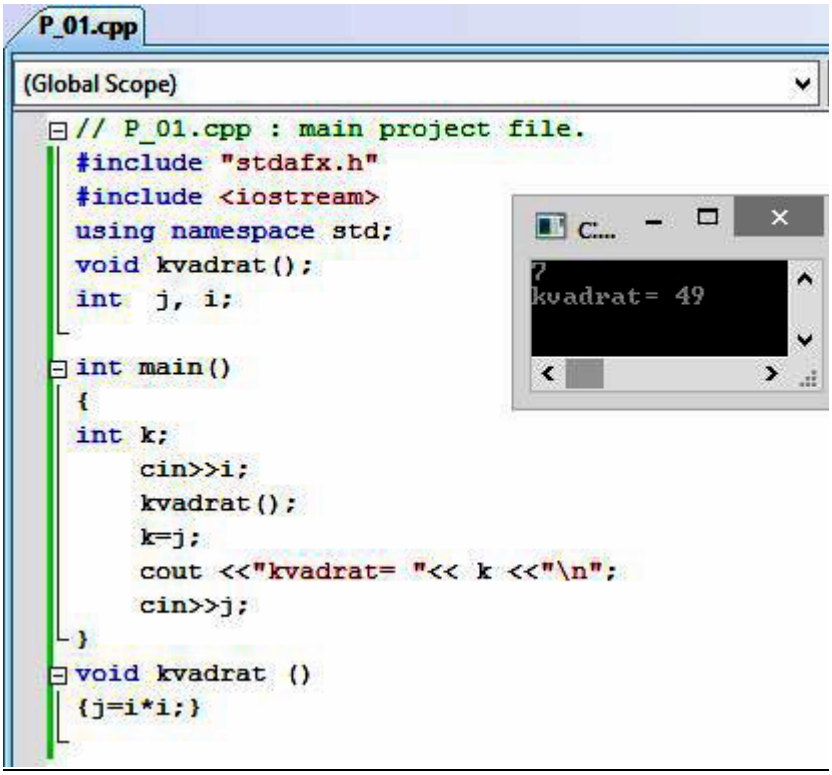


Рис. 3.38 - Листинг программы с внешними переменными

Внешние переменные могут задаваться и не в начале программы. Если переменная используется в функции раньше, чем была объявлена, то она должна быть заявлена в этой функции со словом extern. Пример для этого варианта приведен на рис. 3.39 ниже.

Важно заметить,

1. *объявление* внешней переменной приводит к выделению для нее памяти;
2. *заявление* указывает тип переменной, которая будет использоваться в функции.
3. *область видимости переменной (функции)* – это часть программы, в которой этой переменной можно пользоваться. Для локальных переменных – это функция, где они объявлены. *Область действия внешней* переменной простирается от точки программы, где она объявлена, до конца файла.

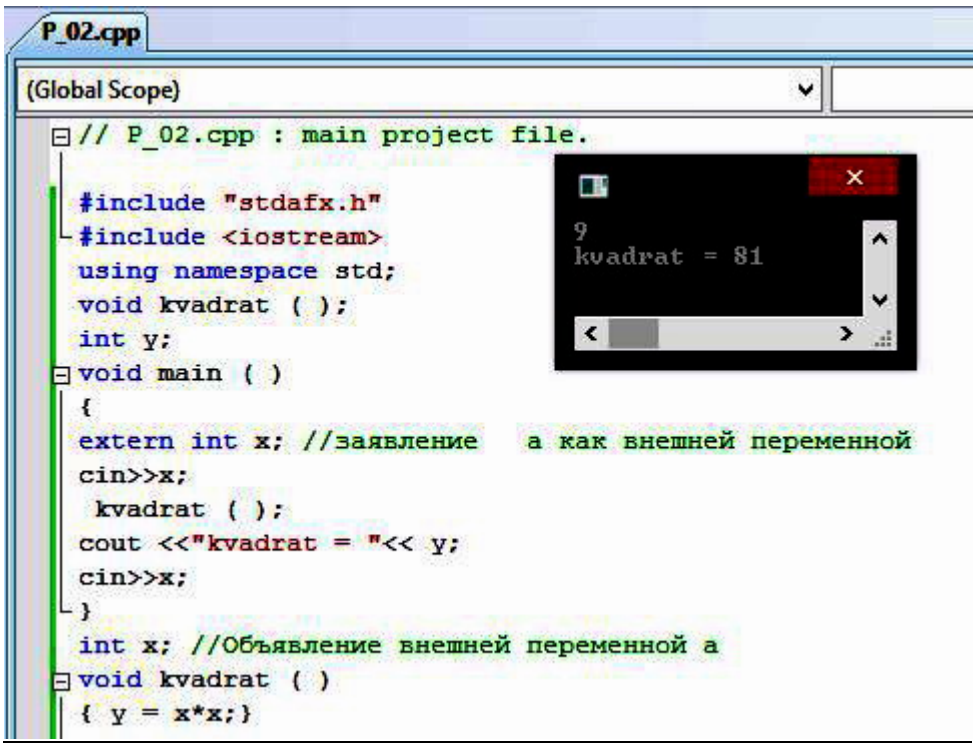


Рис.3.39 - Листинг программы с внешними переменными

Задание 3.18

1. Создайте программу. Главная функция этой программы должна вызывать одну из приведенных функций. В программе использовать внешние переменные.

а) y1=| sin(a*x)/(a*x)|

б) $y2=x^3$

в) y3=a*Sin(a*x)*exp(-x)

г) y4=Tg(ax)* exp(-x)

2. Создайте программу, которая должна:

- записывать в файл 3 числа. Результат записи контролируйте с помощью *«Блокнота»*.

- добавить в файл еще 3 числа.

- прочитать эти 6 чисел и вывести их на экран в виде матрицы 3x2

3. Создайте программу, которая позволяла вводить 10 символов с клавиатуры.

Функция main должна обращаться к дополнительно созданной функции. Эта дополнительная функция с помощью статической переменной должна считать сколько раз при вводе символов нажали клавишу «к».

ЛИТЕРАТУРА

1. Отчет о научно-исследовательской работе. Структура и правила оформления. [Текст]: ГОСТ 7.32-2001; введен 01.07.2002
2. Общие требования к текстовым документам. Единая система конструкторской документации. Межгосударственный стандарт. [Текст]: ГОСТ2.105-95. Взамен ГОСТ 2.105-79, ГОСТ 2.906-71. Введен 01.07.96. Издание (июнь 2002 г.) с поправкой (ИУС 12-2001).
3. Библиографическое описание документа. Общие требования и правила составления. [Текст]: ГОСТ 7.1-84. – Взамен 7.1-76 ; введен 01.01.86
4. Библиографическая запись. Общие требования и правила составления. Система стандартов по информатике, библиотечному и издательскому делу. [Текст]: ГОСТ 7.1-2003. Введен 30.06.2004
5. Библиографическая запись. Заголовок. Общие требования и правила составления. Система стандартов по информации, библиотечному и издательскому делу. [Текст]: ГОСТ 7.80-2000. Введен 01.07.2000.
6. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. Система стандартов по информации, библиотечному и издательскому делу. [Текст]: ГОСТ 7.82-2001. Введен 01.07.2002
7. Диссертация и автореферат диссертации. Структура и правила оформления. Система стандартов по информации, библиотечному и издательскому делу. [Текст]: ГОСТ 7.0.11-2011. Введен (впервые) 13.12.2011 г. № 811-ст.
8. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. Единая система программной документации. [Текст]: ГОСТ 19.701-90 (ИСО 5807-85). Введен 01.01.92
9. Схемы алгоритмов и программ. Обозначения условные графические. Единая система программной документации. [Текст]: ГОСТ 19003-80. Взамен ГОСТ 19428-74. Введен 01.01.92
10. Википедиа [Электронный ресурс] код доступа https://ru.wikipedia.org/wiki/
11. Керриган Б., Ритчи Д. Язык программирования Си. Пер. с англ.. [Электронный ресурс] код доступа http://khpi-iip.mipk.kharkiv.edu/library/pgm/kr/
12. Подбельский В.В. Стандартный Си++:учеб. пособие/М.: Финансы и статистика, 2014, 688с.
13. Пахомов Б.И. С/С++ и MS Visual C++ 2010 для начинающих. – СПб.:БХВ - Петербург, 2012 -736с.
14. Programming Languages – C++. International Standard. ISO/IEC 14822. Second Additional, 2003-10-15^ ISO/IEC/ANSI/ITI? 2003.
15. ЛитвиненкоН.А. Технология программирования на С++. Начальный курс.- СПб.: БХВ-Петербург, 2005. -288с.

16. Литвиненко Н. А.Технология программирования на C++. Win32 API-приложения. —СПб.: БХВ-Петербург, 2010. — 288 с.: ил. — (Учебное пособие). [Электронный ресурс] код доступа http://www.proklondike.com/books/cpp/technology_of_programming_on_cplus.html

УНИВЕРСИТЕТ ИТМО

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА ЭКОНОМИКИ И СТРАТЕГИЧЕСКОГО МЕНЕДЖМЕНТА

Кафедра экономики и стратегического менеджмента с 9 февраля 2015г. является приемником кафедры прикладной экономики и маркетинга.

Кафедра прикладной экономики и маркетинга была основана 25 мая 1995 года в связи с началом подготовки в СПбГУ ИТМО бакалавров по направлению 521600 «Экономика». В 1997 году кафедра стала готовить сначала бакалавров, а затем и специалистов по специальности 071900 «Информационные системы в экономике». Со дня основания и по настоящее время кафедрой руководит Почётный работник высшего профессионального образования, доктор экономических наук, профессор, действительный член Российской академии естествознания Олег Валентинович Васюхин.

В настоящее время кафедра прикладной экономики и маркетинга обучает студентов по специальности 080801 «Прикладная информатика в экономике», а также готовит бакалавров и магистров по направлению 080100 «Экономика»

Кадровый состав кафедры представлен специалистами высшей квалификации – три доктора экономических наук, профессора; 9 кандидатов наук, доцентов и 6 старших преподавателей и ассистентов. Около 30% преподавателей – это молодые специалисты, обучающиеся в аспирантуре или недавно закончившие её.

В соответствии с утверждёнными учебными планами, преподаватели кафедры читают более 40 экономико-управленческих и информационных дисциплин как для студентов своих специальностей и направлений, так и для студентов всего университета. С целью обеспечения более эффективного учебного процесса преподавателями кафедры разработаны более 25 учебно-методических пособий, в том числе, часть из них в виде электронных учебников.

Кафедра обладает современной материально-технической базой. Большая часть учебного процесса реализуется в компьютерных классах факультета технологического менеджмента и инноваций, подключённых к сети Интернет. Все виды занятий, текущий контроль знаний, а также

разнообразные виды самоподготовки студентов осуществляются на основе балльно-рейтинговой системы организации учебного процесса.

Ежегодно кафедра выпускает около 50 специалистов, бакалавров и магистров, которые успешно работают на предприятиях различных форм собственности и направлений деятельности. Часть выпускников каждый год продолжают обучение в аспирантуре СПбГУИТМО. Практически все студенты кафедры, начиная с 3-4 курса, в свободное время работают на предприятиях Санкт-Петербурга, что в большинстве случаев является основой для прохождения различного рода практик и подготовки выпускной квалификационной работы.

Преподаватели и аспиранты кафедры ведут активную научно-исследовательскую деятельность, участвуя в хоздоговорных исследованиях для предприятий и организаций, а также в крупных госбюджетных НИР. В учебном процессе кафедры принимают участие представители промышленности и науки Санкт-Петербурга.

В настоящее время кафедра экономики и стратегического менеджмента факультета технологического менеджмента и инноваций (бывшая кафедра прикладной экономики и маркетинга) является одной из ведущих выпускающих кафедр СПбГУ ИТМО.

Вадим Юрьевич Петров

Информатика. Алгоритмизация и программирование.

Учебное пособие

В авторской редакции
Дизайн В.Ю.Петров
Верстка В.Ю.Петров
Редакционно-издательский отдел Санкт-Петербургского государственного университета информационных технологий, механики и оптики
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати _____
Заказ № _____
Тираж 100
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО

197101, Санкт-Петербург, Кронверкский пр., 49