**Московский государственный технический университет**

**им. Н.Э. Баумана.** **Факультет «Информатика и управление»**

**Кафедра «Системы обработки информации и управления»**

**Методические указания**

**к лабораторным работам по курсу**

**«Основы программирования»**

**(1 семестр)**

Москва, 2018 г.

# Оглавление

[Введение 7](#_Toc409125603)

[Рекомендации по выполнению практической части лабораторных работ 7](#_Toc409125604)

[Методы процедурного программирования 8](#_Toc409125605)

[1. Модульное проектирование 8](#_Toc409125606)

[2. Структурное программирование 9](#_Toc409125607)

[2.1. Проектирование сверху вниз 10](#_Toc409125608)

[2.2. Модульное программирование 11](#_Toc409125609)

[3. Структурное кодирование 12](#_Toc409125610)

[Лабораторная работа 1. Создание, компиляция и отладка программ на С++ в MS Visual Studio. 16](#_Toc409125611)

[1. Цель и задачи ЛР 16](#_Toc409125612)

[2. Порядок выполнения работы 16](#_Toc409125613)

[3. Интегрированная среда разработки Microsoft Visual C++ 16](#_Toc409125614)

[3.1 Запуск IDE. Типы приложений 16](#_Toc409125615)

[3.2 Создание нового проекта 17](#_Toc409125616)

[3.3 Добавление к проекту файлов с исходным кодом 18](#_Toc409125617)

[Добавление нового файла 18](#_Toc409125618)

[Добавление существующего файла 19](#_Toc409125619)

[3.4 Многофайловые проекты 19](#_Toc409125620)

[3.5 Компиляция, компоновка и выполнение проекта 19](#_Toc409125621)

[Конфигурация проекта 21](#_Toc409125622)

[Как открыть проект, над которым вы работали ранее 21](#_Toc409125623)

[4. Встроенная справочная система 21](#_Toc409125624)

[5. Проблемы с вводом-выводом кириллицы 21](#_Toc409125625)

[5.1. Замечания по потоковому вводу-выводу 22](#_Toc409125626)

[6 Работа с отладчиком 23](#_Toc409125627)

[6.1 Установка точки прерывания 24](#_Toc409125628)

[6.2 Выполнение программы до точки прерывания 24](#_Toc409125629)

[6.3 Пошаговое выполнение программы 24](#_Toc409125630)

[6.3.1 Проверка значений переменных во время выполнения программы 24](#_Toc409125631)

[Окна Auto и Watch 25](#_Toc409125632)

[7 Содержание отчета по лабораторной работе 26](#_Toc409125633)

[8. Контрольные вопросы 26](#_Toc409125634)

[9 Рекомендуемые источники информации 26](#_Toc409125635)

[Лабораторная работа 2. Программирование разветвляющихся алгоритмов 27](#_Toc409125636)

[1. Цель работы 27](#_Toc409125637)

[2. Задание 27](#_Toc409125638)

[3. Рекомендации по разработке программы 27](#_Toc409125639)

[4. Требования к отчету 30](#_Toc409125640)

[5. Контрольные вопросы 31](#_Toc409125641)

[6. Рекомендуемые источники информации 31](#_Toc409125642)

[Лабораторная работа 3. Табулирование функций с использованием рядов Тейлора 33](#_Toc409125643)

[1. Цель работы 33](#_Toc409125644)

[2. Задание 33](#_Toc409125645)

[3. Рекомендации по выполнению работы 33](#_Toc409125646)

[3.1 Указание к задаче 1 задания 33](#_Toc409125647)

[3.2 Указание к задаче 2 задания 33](#_Toc409125648)

[3.3 Указание к задаче 3 задания 33](#_Toc409125649)

[3.4 Указание к задаче 4 задания 34](#_Toc409125650)

[4. Содержание отчета. 34](#_Toc409125651)

[5 Контрольные вопросы 34](#_Toc409125652)

[6. Рекомендуемые источники информации 35](#_Toc409125653)

[7. Варианты задания 35](#_Toc409125654)

[Вариант 1 35](#_Toc409125655)

[Вариант 2 35](#_Toc409125656)

[Вариант 3 36](#_Toc409125657)

[Вариант 4 37](#_Toc409125658)

[Вариант 5 37](#_Toc409125659)

[Вариант 6 38](#_Toc409125660)

[Вариант 7 38](#_Toc409125661)

[Вариант 8 39](#_Toc409125662)

[Вариант 9 39](#_Toc409125663)

[Вариант 10 40](#_Toc409125664)

[Вариант 11 40](#_Toc409125665)

[Вариант 12 41](#_Toc409125666)

[Вариант 13 41](#_Toc409125667)

[Лабораторная работа 4 Численные методы решения нелинейных уравнений………………………. 42](#_Toc409125668)

[1. Цель работы. 42](#_Toc409125669)

[2. Задание. 42](#_Toc409125670)

[3. Рекомендации по выполнению работы 43](#_Toc409125671)

[4. Содержание отчета 43](#_Toc409125672)

[5. Контрольные вопросы 44](#_Toc409125673)

[6. Рекомендуемые источники информации 44](#_Toc409125674)

[6. 45](#_Toc409125675)

[Лабораторная работа 5 Сортировка одномерного числового массива………………………………… 45](#_Toc409125676)

[1. Цель работы 45](#_Toc409125677)

[2. Задание 45](#_Toc409125678)

[3. Рекомендации по выполнению работы 45](#_Toc409125679)

[4. Примеры работы с массивами 46](#_Toc409125680)

[4.1 Количество элементов между минимумом и максимумом 46](#_Toc409125681)

[4.2 Динамические массивы 49](#_Toc409125682)

[4.3 Использование датчика случайных чисел. 50](#_Toc409125683)

[5 Содержание отчета 50](#_Toc409125684)

[6 Контрольные вопросы 50](#_Toc409125685)

[7 Рекомендуемые источники информации 50](#_Toc409125686)

[Лабораторная работа 6. Численное интегрирование функций 52](#_Toc409125687)

[1. Цель работы. 52](#_Toc409125688)

[2. Задание. 52](#_Toc409125689)

[3. Рекомендации по выполнению работы. 52](#_Toc409125690)

[3.1. Метод прямоугольников*.* 53](#_Toc409125691)

[3.2. Метод трапеций. 54](#_Toc409125692)

[3.3. Формулы для вычисления точных значений интеграла: 54](#_Toc409125693)

[3.4. Примеры передачи в функцию в качестве параметров одномерных массивов и имен функций. 54](#_Toc409125694)

[*3.5.* Пример вывода таблицы результатов 56](#_Toc409125695)

[3.6 Функция для печати таблицы результатов 57](#_Toc409125696)

[4. Содержание отчета 57](#_Toc409125697)

[5. Контрольные вопросы 57](#_Toc409125698)

[6. Рекомендуемые источники информации 57](#_Toc409125699)

[Лабораторная работа 7 Обработка и печать числовой матрицы 58](#_Toc409125700)

[1. Цель работы 58](#_Toc409125701)

[2. Задание 58](#_Toc409125702)

[3. Рекомендации по выполнению работы 59](#_Toc409125703)

[3.1 Создание двухмерных динамических массивов 59](#_Toc409125704)

[3.2 Передача многомерного массива в функцию с помощью параметров. 60](#_Toc409125705)

[3.3 Пример разработки программы сортировки строк матрицы 61](#_Toc409125706)

[3.4 Основные правила работы с двухмерными массивами 63](#_Toc409125707)

[3.5 Рекомендации по создания программы 64](#_Toc409125708)

[4. Содержание отчета 64](#_Toc409125709)

[5. Контрольные вопросы 64](#_Toc409125710)

[6. Рекомендуемые источники информации 64](#_Toc409125711)

[Лабораторная работа 8 Шифрование текстовых файлов 65](#_Toc409125712)

[1. Цель работы 65](#_Toc409125713)

[2. Задание 65](#_Toc409125714)

[3. Рекомендации по выполнению работы 66](#_Toc409125715)

[3.1. Ввод-вывод строк 66](#_Toc409125716)

[3.2. Пример программы работы с символьными строками. 68](#_Toc409125717)

[3.3. Работа с файлами 72](#_Toc409125718)

[3.4. Потоки ввода-вывода. 77](#_Toc409125719)

[Функции для обмена с потоками 78](#_Toc409125720)

[Функции чтения 79](#_Toc409125721)

[3.5. Использование аргументов командной строки 81](#_Toc409125722)

[4. Содержание отчета 83](#_Toc409125723)

[5. Контрольные вопросы 83](#_Toc409125724)

[6. Рекомендуемые источники информации 83](#_Toc409125725)

[Лабораторная работа 9 Вычисление обратной матрицы методом Гаусса-Жордана…………………….. 84](#_Toc409125726)

[1. Цель работы 84](#_Toc409125727)

[2. Задание 84](#_Toc409125728)

[3. Рекомендации по выполнению работы 85](#_Toc409125729)

[3.1. Алгоритм вычисления обратной матрицы 85](#_Toc409125730)

[Шаг 1. Прямой ход 87](#_Toc409125731)

[Шаг 2. Обратный ход 87](#_Toc409125732)

[*3.2.* *Точность вычисления обратной матрицы.* 87](#_Toc409125733)

[4. Содержание отчета 88](#_Toc409125734)

[5. Контрольные вопросы 88](#_Toc409125735)

[6. Рекомендуемые источники информации 88](#_Toc409125736)

[Лабораторная работа 10 Ввод, сортировка и двоичный поиск в массиве структур…………………. 89](#_Toc409125737)

[1. Цель и задачи работы 89](#_Toc409125738)

[2. Задание и требования к результатам работы 89](#_Toc409125739)

[3. Рекомендации по выполнению работы 89](#_Toc409125740)

[6.1. Шаги разработки программы 89](#_Toc409125741)

[3.2 Работа со структурами 91](#_Toc409125742)

[3.3 Дополнительные требования для «сильных» студентов: 92](#_Toc409125743)

[4. Содержание отчета 92](#_Toc409125744)

[5. Контрольные вопросы 92](#_Toc409125745)

[6. Рекомендуемые источники информации 92](#_Toc409125746)

[Домашнее задание. Методические указания к домашнему заданию по курсу «Основы программирования» 93](#_Toc409125747)

[1. Цели домашнего задания 93](#_Toc409125748)

[2. Требования к выполнению задания 93](#_Toc409125749)

[2.1 Групповая разработка проектов 94](#_Toc409125750)

[2.2 Шаги выполнения задания 95](#_Toc409125751)

[3 Требования к отчету 97](#_Toc409125752)

[4 Оценка выполнения задания 97](#_Toc409125753)

[5. Варианты заданий 97](#_Toc409125754)

[Вариант 1 97](#_Toc409125755)

[Вариант 2 98](#_Toc409125756)

[Вариант 3 98](#_Toc409125757)

[Вариант 4 98](#_Toc409125758)

[Вариант 5 98](#_Toc409125759)

[Вариант 6 98](#_Toc409125760)

[Вариант 7 98](#_Toc409125761)

[Вариант 8 98](#_Toc409125762)

[Вариант 9 98](#_Toc409125763)

[Вариант 10 98](#_Toc409125764)

[Вариант 11 98](#_Toc409125765)

[Вариант 12 98](#_Toc409125766)

[Вариант 13 98](#_Toc409125767)

[Вариант 14 99](#_Toc409125768)

[Вариант 15 99](#_Toc409125769)

[Вариант 16 99](#_Toc409125770)

[Вариант 17 99](#_Toc409125771)

[Вариант 18 99](#_Toc409125772)

[Вариант 19 99](#_Toc409125773)

[Вариант 20 99](#_Toc409125774)

[Вариант 21 99](#_Toc409125775)

[Вариант 22 99](#_Toc409125776)

[Вариант 23 99](#_Toc409125777)

[Вариант 24 99](#_Toc409125778)

[Вариант 25 99](#_Toc409125779)

[6 Вспомогательные материалы для выполнения лабораторных работ 100](#_Toc409125780)

# Введение

Цикл лабораторных работ по курсу «Основы программирования» выполняется студентами кафедры ИУ5, обучающимися по программе бакалавриата, в 1-ом семестре.

Цикл включает 10 аудиторных лабораторных работ + 1 работа – домашнее задание.

Назначение цикла ЛР - приобретение студентами практических умений и навыков процедурного программирования на языке C++ .

ЛР проводятся в компьютерных классах на ПК в среде программирования MS Visual Studio.

Процесс проведения лабораторных работ включает подготовительную, практическую и заключительную части.

*Подготовительная часть* выполняется до проведения аудиторных занятий. Студент должен изучить соответствующий раздел теоретического курса и методических указаний , уяснить цель работы и методику ее проведения.

*Практическая часть* является основным структурным элементом лабораторной работы. В ходе практической части студенты самостоятельно ***разрабатывают*** и ***тестируют*** программу на языке С++, позволяющую выполнить требования задания и получают данные для составления отчета по работе.

После завершения практической части студент должен продемонстрировать преподавателю правильность выполнения пунктов задания лабораторной работы. Программа должна обеспечивать ***удобство изменения*** студентом ***входных данных*** в процессе проверки преподавателем отдельных пунктов задания.

*Заключительная часть* лабораторной работы включает анализ результатов, формулировку выводов по результатам выполнения заданий, представление отчёта по работе и ответы на вопросы.

Качество выполнения пунктов задания влияет на оценку за выполнение работы.

# Рекомендации по выполнению практической части лабораторных работ

Цель лабораторных занятий – приобретение студентами практических навыков выполнения основных этапов разработки программы методами процедурного программирования. Основными этапами разработки программы являются:

* постановка задачи (проектирование программы),
* разработка алгоритма (структурное программирование),
* кодирование и отладка (структурное кодирование),
* тестирование.

## Постановка задачи

*Этап постановки задачи* заключается в согласовании с преподавателем задания на выполнение лабораторной работы. Текст задания естественно может по-разному пониматься студентом и преподавателем. Учитывая, что работу выполняют студенты 1-го курса, на первых занятиях преподаватель требует от студента объяснения того, что он намеривается сделать в данной работе. Это позволит избежать больших потерь времени у студентов на не нужную работу. В дальнейшем студенты должны сами, в случае необходимости, инициировать согласование задания с преподавателем.

## Разработка алгоритма

*Разработка алгоритма* является наиболее творческим и важным этапом разработки программы. Однако большинство студентов сразу начинают с кодирования, а затем по коду программы рисуют подробную, с точностью до оператора, схему алгоритма. Чтобы дать понять студентам необходимость этапа разработки алгоритма, уже во 2-й работе им предлагается интуитивно понятная, но логически более сложная по сравнению с расчетом по формулам, задача, решение которой без предварительной разработки алгоритма у большинства студентов вызывает трудности.

Мировой опыт практической алгоритмизации и программирования привел к формированию неких общих принципов и методов проектирования, разработки и оформления алгоритмов.

Какими качествами должен обладать ***хороший*** алгоритм?

Прежде всего, от алгоритма требуется, чтобы он правильно решал поставленную задачу. Но не менее важно, какой ценой это достигается. Речь идет о разумности затрат на его создание. С этой точки зрения алгоритм должен быть легким для понимания, простым для доказательства правильности и удобным для модификации. В рамках такой идеологии и сформировался так называемый структурный подход к конструированию и оформлению алгоритмов, позволяющий уменьшить количество ошибок в алгоритмах, упростить контроль, модификацию и сократить время разработки программы.

Основная идея структурного подхода – разделение алгоритма на структурные блоки небольшого размера, каждый из которых имеет ***один*** вход и ***один*** выход. Каждый блок выполняет определённую функцию и имеет небольшой размер по сравнению с размером всей программы, благодаря чему его можно протестировать независимо от остальных блоков программы.

По своей сути структурный подход есть отказ от беспорядочного стиля в алгоритмизации и программировании (в частности, отказ от оператора goto) и определение ограниченного числа стандартных приемов построения легко читаемых алгоритмов и программ с ясно выраженной структурой. Теоретическим фундаментом этого подхода является теорема о структурировании, из которой следует, что алгоритм решения любой практически вычислимой задачи может быть представлен с использованием трех элементарных базисных управляющих структур:   
а) структуры следования или последовательности;   
б) структуры ветвления;   
в) структуры цикла с предусловием.

В процессе выполнения ЛР студенты должны освоить методику структурного программирования, которая предполагает *пошаговую детализацию* алгоритма:  
сначала разрабатывается и проверяется на тестовом примере укрупненная структурная схема алгоритма, а затем выполняется *последовательное* уточнение его структурных блоков.

## Кодирование

*На этапе кодирования* и отладки программы предлагается также пользоваться пошаговой детализацией кода программы.   
На первых занятиях студенты обычно пишут код всей программы и пытаются выполнить ее отладку, не разрабатывая контрольных примеров для отдельных блоков. Часто отсутствие синтаксических ошибок в коде считается успешным завершением разработки программы. В действительности устранение синтаксических ошибок занимает незначительную часть времени разработки программы по сравнению с поиском и исправлением логических ошибок в алгоритме решения задачи.

Для эффективного устранения ошибок требуется отладка программы по блокам. Такой подход предполагает, что после каждого шага детализации программа должна быть работоспособной (должна «дышать»). Сначала разрабатывается и выполняется скелет программы, а затем в программу последовательно добавляются и отлаживаются блоки алгоритма. Такой подход существенно сокращает время отладки программы, так как ошибки локализованы в добавленном блоке.

Для реализации «структурного» метода программирования блоки должны удовлетворять определенным требованиям, возможность выполнения которых формируется на этапе разработки алгоритма.

***Базисный набор*** управляющих структур является функционально полным, то есть с его помощью можно создать любой сколь угодно сложный алгоритм. Однако с целью создания более компактных и наглядных алгоритмов дополнительно используются следующие управляющие структуры: а) структура сокращенного ветвления; б) структура выбора; в) структура цикла с параметром; г) структура цикла с постусловием. В разных языках программирования реализация базовых управляющих структур может быть различной. В языке C++ реализованы все перечисленные структуры.

Любой алгоритм может быть построен посредством композиции базисных и дополнительных структур путём их последовательного соединения (образование последовательных конструкций) и их вложения друг в друга (образование вложенных конструкций).

Процесс программирования − это запись разработанного алгоритма на специальном языке (языке программирования) − представление алгоритма на языке, "понятном" исполнителю (транслятору), т. е. в форме, допускающей ввод в машину и последующий перевод на машинный язык (в коды машины).

Язык программирования − это строго формализованный язык для описания процесса решения задачи на ЭВМ, который представляет собой совокупность ограниченного набора символов и строгих правил их использования. Составленная программа вводится в ЭВМ и затем автоматически переводится на язык машины с помощью специальных программных средств, позволяющих автоматизировать этот процесс. Перевод – "трансляция" исходного текста программы выполняется служебной программой – транслятором, который осуществляет синтаксический контроль текста программы и последующий его перевод.

Трансляторы могут быть компилирующего типа – компиляторы и интерпретирующего типа – интерпретаторы.

Компилятор анализирует и преобразует исходный текст в так называемый объектный код (промежуточное состояние программы в относительных адресах и с неразрешенными внешними ссылками) с использованием всей логической структуры программы. Затем программа, представленная в объектном коде, обрабатывается служебной программой – компоновщиком, который осуществляет подключение внешних подпрограмм (разрешение внешних ссылок) и выполняет дальнейший перевод программы пользователя в коды машины (в загрузочный код с относительной адресацией машинных команд, начиная от начала программы). Программа в загрузочном коде сохраняется в .exe-файле и может быть выполнена на компьютере. Загрузка программы из .exe-файла в память машины для её выполнения осуществляется служебной программой загрузчик.

Интерпретатор сразу производит анализ, перевод (в машинный код) и выполнение программы строка за строкой. Поэтому интерпретатор должен находиться в оперативной памяти в течение всего времени выполнения программы пользователя. При интерпретации скорость выполнения программы существенно снижается, однако весь процесс прохождения программы на ЭВМ упрощается и имеется возможность организации диалогового (интерактивного ) режима отладки и выполнения программы.

Выбор языка программирования определяется многими факторами: типом решаемой задачи, располагаемыми вычислительными средствами, вкусами и знаниями заказчика и разработчика.

## Тестирование программы

Отладка программы – это процесс поиска и устранения ошибок. Часть ошибок формального характера, связанных с нарушением правил записи конструкций языка или отсутствием необходимых описаний, обнаруживает транслятор, производя синтаксический анализ текста программы. Транслятор выявляет ошибки и сообщает о них, указывая их тип и место в программе. Такие ошибки называются ошибками времени трансляции или *синтаксическими ошибками*.

Ошибочные ситуации могут возникнуть и при выполнении программы, например, деление на нуль или извлечение корня квадратного из отрицательного числа. Такие ошибки называются *ошибками времени выполнения*.

Программа, не имеющая ошибок трансляции и выполнения, может и не дать верных результатов из-за логических ошибок в алгоритме, т. е. *алгоритмических, или семантических ошибок*. Ошибки подобного рода могут возникнуть на любом этапе разработки программы: постановки задачи, разработки математической модели или алгоритма. Необходим действенный контроль над процессом вычислений, позволяющий предотвращать или своевременно обнаруживать ошибки подобного рода. Для этого используются как качественный анализ задачи, основанный на различного рода интуитивных соображениях и правдоподобных рассуждениях, так и контрольный просчет или тестирование программы.

*Тестирование программы* – это выполнение программы на наборах исходных данных (*тестах*), для которых известны результаты, полученные другим способом. Система тестов подбирается таким образом, чтобы

а) проверить все возможные режимы работы программы (все ветви на схеме алгоритма);

б) по возможности, локализовать ошибку.

При тестировании программы простой и действенный метод дополнительного контроля над ходом её выполнения – создание *контрольных точек*, т. е. контрольный вывод промежуточных результатов.

Для проверки правильности работы программы иногда полезно также выполнить проверку выполнения условий задачи (например, для алгебраического уравнения найденные корни подставляются в исходное уравнение и проверяются расхождения левой и правой частей).

Для сложных по структуре программ плохо спланированные процессы алгоритмизации и программирования приводят к ошибкам, которые могут быть обнаружены лишь после многократных проверок, и процесс отладки и тестирования может потребовать значительно больше машинного времени, чем собственно кодирование.

# Методы процедурного программирования

На протяжении долгого времени в программировании применялся процедурный подход к разработке программ, при котором программа состоит из функций, ***вызывающих друг друга.***

Гораздо позже появилось объектно-ориентированное программирование (ООП), которое позволяет ***группировать функции и данные в единой сущности - «объекте»***. Чтобы   
объектно-ориентированный подход «работал» (упрощал разработку программы), объект должен представлять собой законченную, интуитивно понятную сущность. То есть, объект - это нечто большее, чем просто «группировка» функций и данных.

Нельзя сказать, что объектный подход ***всегда*** лучше процедурного.

Как и любой образ мыслей, объектно-ориентированный подход может быть удобен или не удобен, в зависимости от ситуации.

**Бывают задачи, в которых сложно выделить сущность.** Например, «алгоритм нахождения наибольшего общего делителя». В нём есть два числа и численные операции над ними.

Какие сущности следует оформлять в объект, а какие - нет, это решает разработчик. Приемы структурного программирования, являющиеся неотъемлемой частью процедурного программирования, используются при разработке функций в ООП.

## Модульное проектирование

       Каждому известны два наиболее важных параметра программ: память и время выполнения. Однако существует еще один не менее важный параметр, который необходимо учитывать, — время разработки, которое зависит от сложности программы. Сложность определяется размерностью программы и ее структурой (количеством ветвлений и циклов). Чем сложнее программа, тем труднее ее контролировать, тестировать, отлаживать и вносить изменения в процессе эксплуатации. Ограничив параметр сложности частей программы, мы можем получить намного легче поддающийся управлению процесс разработки надежных программ.

Метод, обычно используемый для управления сложностью, заключается в том, что процесс или структуру разбивают на небольшие, легко управляемые части (модули), которые комбинируют для получения определенной функции. Такой подход используется во всех организациях: военных, административных и промышленных. Например, армия состоит из дивизий, дивизия – из полков, полки – из батальонов, и так далее.

Управление сложностью программы включает следующие шаги:

* модульное ***проектирование*** (составление программы из отдельных модулей);
* структурное ***программирование*** (управление сложностью в пределах каждого модуля);

*Увеличение сложности программы не пропорционально увеличению ее размера.  
 При увеличении размера программы вдвое сложность может возрасти в четыре и более раз.*

Управление сложностью — очень важный фактор в программировании. По существу мы занимаемся этим, определяя функцию каждого модуля независимым образом. Каждый модуль, получая входные данные, должен выдать определенные выходные данные.

***Модули*** оформляются в виде отдельного ***файла***, тестируются независимо от других модулей и могут включаться в программы на С++, например, с помощью оператора #include, что дает возможность составлять из модулей систему с более сложной структурой.

       Следующий шаг в управлении общей сложностью — управление сложностью в пределах каждого модуля; этот процесс обычно называется структурным программированием.

## Структурное программирование

Овладение методами ***структурного программирования*** является одной из основных задач курса лабораторных работ.

       Разбиение программы на части (уровни) позволит нам контролировать её сложность. Детали нижнего уровня могут игнорироваться при программировании текущего уровня, если нет необходимости в соответствующей этому уровню информации, или заменяться «заглушками». Например, если выполняется основная программа и встретился вызов функции, то можно отложить на потом её разработку и вернуть из функции тестовые значения результатов её выполнения (или ограничиться печатью некоторых сообщений) и продолжать следовать логике основной программы.

Структурное программирование сосредоточивается на одном из наиболее подверженных ошибкам факторов программирования — логике программы — и включает три главные составляющие:  
 1. ***Проектирование*** сверху вниз.  
 2. Модульное ***программирование***.  
 3. Структурное ***кодирование***.

Если при разработке программы не соблюдать правила структурного программирования, то вначале программа может иметь довольно ясную структуру, но по мере ее изменений и корректировок в процессе автономной и комплексной отладки, а также после возможных изменений на этапе эксплуатации программы, логика последней становится довольно сложной.

### Проектирование сверху вниз

Проектирование программ сверху вниз подобно написанию статьи сверху вниз. Процесс написания статьи имеет иерархическую структуру и начинается с вершины иерархии, т.е. с краткого обзора. Разработку проекта обычно начинают с исследования целей и определения основных задач, ведущих к достижению этих целей. Если проект очень большой, то необходимо провести его разделение на части, которое должен выполнить компетентный и квалифицированный специалист.

Вначале необходимо написать то, что вы хотите сделать, на естественном языке. Этот шаг часто многое раскрывает. Нередко вы обнаруживаете, что не в состоянии записать задачу на естественном языке. В таком случае не надейтесь, что вам удастся составить программу. И потом ведь намного легче переделать описание задачи на естественном языке в период разработки спецификаций (прототипов функций и типов переменных), чем переписывать потом программу, которую считают уже завершенной.

Таким образом, важно постараться как можно более точно сформулировать задачу на стадии проектирования, чтобы уменьшить количество ее изменений на последующих стадиях программирования и отладки.  
***Сначала опишите алгоритм программы на естественном языке.***

Метод проектирования сверху вниз (метод пошаговой детализации) предусматривает сначала определение задачи в общих чертах, а затем постепенное уточнение структуры путем внесения более мелких деталей. Проектирование представляет собой последовательность шагов такого уточнения. На каждом шаге необходимо выявить основные функции, которые нужно выполнить, т.е. данная задача разбивается на ряд подзадач, пока эти подзадачи не станут настолько простыми, что каждой из них будет соответствовать один модуль.   
 Именно такой традиционный и по существу иерархический подход применяется при создании сложных структур в других областях, например, в серийном производстве автомобиля.   
Затем проект системы может быть проверен и подтвержден моделированием или расчетами. Действие каждого модуля может быть описано одной фразой. Если описание модуля занимает больше одной строки, подумайте о его перепроектировании.  
 Далее следует описать данные, указывая их структуру и основные процессы обработки. Это описание должно включать тщательно отобранные *контрольные примеры*, убедительно демонстрирующие функции системы и их наиболее существенные варианты. Такие примеры будут полезны позже на стадии отладки и тестирования. При описании модуля должны быть описаны его тестовые данные. Тестирование программы неизбежно, поэтому выявление требований к тестированию заранее, на стадии проектирования, является хорошей практикой. *Разрабатывайте тестовые данные заранее.*

Логическая проверка фрагментов программы должна упростить тестирование конечной программы. Чтобы выполнить это “ручное” тестирование, спецификации системы должны быть достаточно точными. При дальнейшем проектировании спецификации могут уточняться. *Пример спецификаций – описания структур данных, назначения и прототипов функций для их обработки*. Непосредственно программирование функций будет выполняться на последующих шагах.

Основное преимущество такого метода работы — то, что он обеспечивает создание документации. Документацию всегда начинают создавать во время разработки спецификаций, но часто результаты этой начальной работы утеряны к моменту формирования окончательной документации. Начинайте разработку документации до начала программирования. Это также способствует улучшению программы. Ведь программист должен более тщательно обдумать структуру, данные и тестирование своей программы, чтобы записать все это на бумаге.

Проектирование должно быть завершено до начала программирования. Как только приступают к программированию, создается психологический барьер, препятствующий дальнейшим улучшениям проекта. Следует сделать несколько итераций проекта прежде, чем начать программирование. В небольших проектах, где этап проектирования является менее решающим, часто программисты приступают к программированию слишком рано. Многие программы вообще не проектируются, а создаются сразу в форме кода.  
 *Прежде чем начать программировать, разработайте проект.*

Цена обнаружения и корректирования ошибок в системе возрастает по мере приближения программы к завершению. Исправление ошибок, найденных во время проектирования, относительно недорого (переделать проект) по сравнению с обнаружением и исправлением ошибок на конечном этапе тестирования (перепрограммировать задачу). Кроме того, поздно найденные ошибки затрагивают многих исполнителей проекта и вызывают необходимость связи между группами (например, между программистами, теми, кто делает документацию, и заказчиками).  
 *Исключайте ошибки с самого начала.* В большинстве случаев ошибки в системе — результат неполных или противоречивых спецификаций проекта. Опытные программисты считают, что на стадии спецификаций проекта вносится до двух третей ошибок. Следовательно, именно здесь должны быть сосредоточены наши главные усилия, так как ошибки при проектировании дороже всего обходятся. *Хорошее программирование не спасет плохой проект.*

### Модульное программирование

Чтобы преуспеть в структурном программировании, программу следует представить в виде модулей. Модульное программирование — это процесс разделения программы на логические части, называемые модулями, и последовательное программирование каждой части. Когда большая единая задача делится на подзадачи, то значительно проще понять и прочесть программу. Это обычный способ управления сложной ситуацией: “Разделяй и властвуй!” Если проведено проектирование всей задачи сверху вниз, то она, естественно, разбивается на подзадачи для возможных модулей. При этом преследуют две цели.   
       1. Необходимо добиться того, чтобы программный модуль был правильным и не зависел от контекста, в котором он будет использоваться.   
       2. Следует стремиться к тому, чтобы из модулей можно было формировать большие программы без каких-либо предварительных знаний о внутренней работе модуля.   
       Прикладные подпрограммы и функции стандартных библиотек — примеры удачных модулей. Мы используем процедуру вычисления функции квадратного корня в любом контексте и не ожидаем побочных эффектов в других частях программы. Нам хотелось бы, чтобы наши собственные модули взаимодействовали таким же образом, не оказывая непредусмотренного влияния друг на друга.

Никто не может точно назвать наилучший размер модуля, но обычно используемое ограничение — около 60 строк. Такая длина программы удобна для восприятия, так как она легко охватывается и запоминается. Это число строк помещается на одной странице, кроме того, его легко прочесть на терминале вычислительной машины. Очевидно, существуют исключения, но тем не менее 60 строк — это удобный размер модуля.

      Модуль должен также проверять аргументы на их принадлежность области определения, как это делается для функции квадратного корня. Когда аргумент функции квадратного корня получает значение, выходящее за пределы области определения, обычно выдается сообщение об ошибке. Это называется ***побочным эффектом****.*Иногда побочный эффект включает код ошибки, который может быть обработан. Таким образом, для модуля должны быть определены   
      1) алгоритм решения задачи;

2) область допустимых входных значений;

3) область возможных выходных значений;

4) возможные побочные эффекты.   
       Если эти четыре объекта надлежащим образом определены для каждого модуля, то результатом будут четко составленные модули, и следовательно, хорошие программы. Однако часто область определения не проверяют, а побочные эффекты не определяют.   
       Следует сделать одно дополнительное замечание по поводу побочных эффектов. Лишь в редких случаях получение побочных эффектов должно прекращать выполнение программы. Вместо этого должна выявляться ошибка и должен происходить возврат к вызывающему модулю. Таким образом, нежелательно, чтобы процедура вычисления функции квадратного корня прекращала вычисления при получении значения, выходящего за пределы области определения (т.е. отрицательного значения). В этом случае должно быть возвращено некоторое значение (часто нуль) вызывающему модулю, предупреждающее об ошибке. Затем ***вызывающий модуль*** должен решить, прекращать или продолжать вычисления. Это гарантирует универсальность программы, так как ошибка идентифицируется и ***вызывающа***я программа может делать то, что необходимо.

      Сегменты программ должны быть составлены следующим образом. Каждый сегмент должен иметь ***один вход*** в начале ***и один выход*** в конце. Если другие сегменты вызываются внутри какого-либо сегмента, то в них входят через начало, выходят через конец и возвращаются обратно в вызывающий сегмент.   
       Главная процедура должна принимать все решения по управлению потоком данных для соответствующих обрабатывающих процедур. Переменные, общие для всех модулей, должны быть определены как часть главной процедуры в области глобальных переменных. Локальные переменные должны использоваться только своими модулями.   
       Каждый сегмент должен иметь как можно меньше ветвей вычислений. Чем меньше размер модуля, тем, как правило, меньше ветвей, которые надо тестировать. Чем проще структура, тем лучше. Каждая процедура должна выполнять только одну логическую задачу. Следует обеспечить невмешательство других сегментов в «кухню» вычислений каждой процедуры. Таким образом, обрабатывающая процедура будет управляться только своим сегментом. *Ни одно решение, принятое за пределами сегмента, не должно определять какое-либо действие внутри сегмента, и, наоборот, ни одно решение внутри сегмента не должно сказываться на действиях вне сегмента*. Четкость программы в целом в большой степени зависит от четкости структуры каждого модуля.   
       Таким образом, каждая процедура является замкнутой. Управление передается от главной процедуры к вызываемой и, когда последняя выполнит свою функцию, она возвратит управление вызвавшей ее главной процедуре. Вызываемая процедура может в свою очередь вызвать другую процедуру, однако возврат всегда будет происходить в процедуру, являющуюся вызывающей для данной процедуры.

Следование правилам модульного программирования позволит значительно снизить затраты на сопровождение программы при необходимости внесения в нее изменений на этапе эксплуатации.

### Структурное кодирование

Человек имеет склонность к последовательным процессам. Мы предпочитаем последовательные чтение и обработку. Поэтому, чем больше разрывов в последовательности, тем труднее нам прослеживать логику программы. После нескольких условных переходов мы можем запутаться в программе со сложным ветвлением.

Структурное кодирование — это метод написания хорошо структурированных программ, который позволяет получать программы, более удобные для тестирования, модификации и использования. Задача по созданию программы из базисных структур относится к полю деятельности программиста. Если он не в состоянии сделать это, подлежит сомнению его профессиональная компетенция.

Структурное кодирование состоит в получении правильной программы из некоторых простых логических структур. Оно базируется на строго доказанной теореме о структурировании, которая утверждает, что любую правильную программу (с одним входом и одним выходом, без зацикливаний и недостижимых команд) можно написать с использованием только следующих логических структур:

1) *последовательности* двух или более операторов

2) *выбора* одной из двух последовательностей операторов (ветвление) (IF ELSE)

3) *повторения* последовательности операторов, пока выполняется некоторое условие (цикл WHILE)



Рис.1. Основные логические структуры (a — последовательность, b — выбор и c—повторение).

       На рис.1 показаны эти основные логические структуры. Комбинации правильных программ, полученные с использованием трех основных управляющих структур (последовательности, выбора и повторения), также являются правильными программами. Заметьте, что ***каждая структура имеет один вход и один выход***. Можно получить программу любого размера и сложности, применяя итерацию и вложение этих основных структур. При использовании только указанных трех структур отпадает необходимость в безусловных переходах и метках. Каждая структура прослеживается от начала до конца без каких-либо ветвлений. Применение структурного программирования в значительной степени уменьшает сложность программ.   
       Структура ***последовательность***— формализация того, что операторы программы выполняются в порядке их появления в программе, пока что-то не изменит эту последовательность. Тот факт, что ***последовательность*** является управляющей логической структурой, иногда упускается из виду.

Структура ***выбор***— это выбор одного из двух действий, исходя из выполнения некоторого условия; ей соответствует оператор IF ELSE.

Структура ***повторение***реализует повторное выполнение группы операторов до тех пор, пока выполняется некоторое условие; ей соответствуют оператор WHILE.   
       Существенным моментом является то, что при замене любого функционального прямоугольника в схеме правильной программы на любую из трех основных структур программа остается в классе правильных программ.



Рис. 2. Структура последовательность.

На рис. 2 приведена схема программы, состоящей из последовательности трех функций. Если мы хотим изменить функцию В и использовать в ней структуру выбора, то получим схему, изображенную на рис. 3. Получение правильной программы путем замены операторов программы на управляющие логические структуры называется *вложением*структур.



Рис. 3. Структура с вложением

       Хотя теоретически возможно написать все правильные программы, используя только три перечисленные выше основные логические структуры, мы увидим, что небольшое расширение этого базиса облегчит программирование. Введенные структуры должны иметь один вход и один выход. Обычно добавляют операторы DO WHILE и SWITCH.

На рис. 4 показаны эти две логические структуры.

  
Рис. 4. Операторы do-while и switch.

       Оператор ***do-while*** организует цикл, подобный циклу оператора ***while***. Но существуют различия между этими операторами.   
       1. Цикл ***while*** оканчивается, когда условие ложно, а цикл ***do-while*** — когда условие истинно.   
       2. В цикле ***do-while*** условие проверяется после выполнения операторов цикла, так что операторы тела цикла выполнятся по крайней мере один раз.   
       Структура ***switch*** — это схема разветвления и соединения ветвей, которая используется для выбора одной из ветвей вычисления в зависимости от значения целого выражения (ключа).   
Чтобы программу можно было бы читать сверху вниз как печатный текст,

при написании программы рекомендуется руководствоваться следующими принципами:

- логическая структура должна указываться *отступами* так, чтобы логические отношения в программе соответствовали позициям в листинге;   
- программа должна быть разделена на группы команд (модули), которые легко охватываются;   
- программа должна быть организована таким образом, чтобы ее можно было читать от начала до конца без прерываний, обусловленных переходами. 

# Лабораторная работа 1. Создание, компиляция и отладка программ на С++ в MS Visual Studio.

## Цель работы

Целью работы является приобретение студентами навыков выполнения процессов создания, компиляции и отладки программ на С++ в MS Visual Studio.

Задачи, решаемые при выполнении лабораторной работы:

* Ознакомление с интегрированной средой разработки Microsoft Visual C++.
* Выполнение процессов компиляции, сборки и исполнения программы.
* Размещение файлов программы.
* Структура программы на С++, подключение библиотек.
* Работа с текстом демонстрационной программы.
* Ознакомление с потоковым вводом/выводом.
* Ввод/вывод текста на русском языке.
* Отладка демонстрационной программы. Работа с отладчиком.

Планируемое время выполнения работы- 4 часа.

## Порядок выполнения работы

1. Прочитайте текст и выполните все приведенные в нем примеры.

2. Выполните задание, указанное в п. 3.6.

3. Выполните задания 1 и 2, указанные в конце п. 3.8.

## Интегрированная среда разработки Microsoft Visual C++

*Integrated Development Environment* (интегрированная среда разработки), или, со­кращенно, *IDE* — это программный продукт, объединяющий текстовый редактор, компилятор, отладчик и справочную систему. Предполагается, что пакет Micro­soft Visual Studio, в состав которого входит IDE Microsoft Visual C++, уже установлен на вашем компьютере. Здесь приводятся минимально необ­ходимые сведения для начала работы с интегрированной средой. Более подроб­ную информацию можно извлечь из справочной системы Visual Studio. Если справочная система MSDN не установлена, то к ней можно обратиться через Internet.

Любая программа, создаваемая в среде Visual C++, даже такая простая, как «Hello, World!», всегда оформляется как отдельный *проект (project).* Проект — это набор взаимосвязанных исходных файлов и, возможно, включаемых (заголовочных) файлов, компиляция и компоновка которых позволяет создать исполняемую про­грамму. Однако, разработчики Visual Studio пошли еще дальше, стремясь удов­летворить потребности не только программистов-одиночек, но и больших коллек­тивов разработчиков программных продуктов. Так появилось понятие *рабочей области проекта (Solution).* Рабочая область может содержать несколько различных проектов, сгруппированных вместе для согласованной разра­ботки. По умолчанию папка для хранения проектов *рабочей области* создается компилятором в каталоге c:/Мои документы/Visual Studio /Projects/Имя\_проекта, а в ней создается еще одна папка с именем *Имя\_проекта*, которая будет содержать файлы разрабатываемого проекта.

Внимание!!! При работе в компьютерном классе студенты не имеют прямого доступа к диску ***С:.*** Им выделяется отдельная область дисковой памяти, где они могут хранить свои проекты.

### Запуск IDE. Типы приложений

Вызов Visual C++ осуществляется или через меню Пуск ► Программы ► Microsoft Visual Studio ► Microsoft Visual C++ ( На вашем компьютере путь к исполняемой команде меню может быть другим.), или щелчком мышью по пиктограмме с соот­ветствующим именем, если вы позаботились о ее размещении на рабочем столе компьютера. Запустите Visual C++ на своем компьютере. После запуска Visual C++ появляется главное окно программы.

Под заголовком главного окна, как и во всех Windows-приложениях, находится строка меню. Назначение команд меню и кнопок панелей инструментов мы будем рассматривать по мере необходимости, разбирая основные приемы работы в IDE. Пока же только заметим, что для кнопок панелей инструментов предусмотрена удобная контекстная помощь: если вы наведете курсор мыши на кнопку и задер­житесь на секунду-другую, то всплывет подсказка с назначением кнопки.

Рабочий стол Visual C++ включает в себя три окна:

Окно Solution Explorer *(окно рабочей области)* предназначено для оказания помощи при написании и сопровождении многофайловых программ. Пока что оно пустое, но после создания нового проекта (или загрузки сохраненно­го ранее проекта) одна из вкладок этого окна будет содержать список файлов проекта.

Окно Class View *-* окно просмотра классов *(в* *первом семестре мы его не будем использовать)*.

Окно Output *(окно вывода)* расположено внизу рабочего стола (может быть свернуто) ислужит для вывода сообщений о ходе компиляции, сбор­ки и выполнения программы. В частности, сообщения о возникающих ошибках появляются именно в этом окне.

Visual Studio позволяет строить проекты разных типов, ориентированные на различные сферы применения. Так как эта студия спроектирована на Windows-платформе, то почти все типы проектов являются оконными Windows-приложе­ниями с соответствующим графическим интерфейсом. В то же время разработчики Visual Studio предусмотрели работу и с так называемыми *консольными прило­жениями.*

При запуске консольного приложения операционная система создает так называемое *консольное окно,* через которое идет весь ввод-вывод программы. Внеш­не это напоминает работу в операционной системе MS DOS или других операци­онных системах в режиме *командной строки.* Этот тип приложений больше всего подходит для целей изучения языка C/C++, так как компилируемые программы не «покрываются» толстым слоем промежуточного Windows-кода. В лабораторных работах при изучении языка С++ вы будете создавать ***консольные приложения***.

### Создание нового проекта

Для создания нового проекта типа «консольное приложение» выполните следу­ющие действия:

Выберите в строке меню главного окна команду File ► New...

В открывшемся диалоговом окне New выберите вкладку Projects:

1. выберите Project type: Win32, Templates: Win32 Console Application;
2. введите имя проекта в текстовом поле Project Name, например *Lab1\_1;*
3. введите имя решения в текстовом поле Solution Name, например *Lab1;*
4. введите имя ***каталога размещения файлов проекта*** в текстовом поле Name (если указанный вами каталог отсутствует, то он будет создан автоматиче­ски);
5. щелкните левой кнопкой мыши на кнопке ОК.

После щелчка запускается так называемый *мастер приложений* Application  
Wizard, который открывает диалоговое окно Win32 Console Application   
с предложением определиться, какой подтип консольного приложения вам требуется создать:

1. выберите тип ***An empty project***;
2. щелкните на кнопке ***Finish***.

Допустим, что в качестве Project Name вы ввели имя Lab1\_1.

Прежде чем продолжать работу, свернем временно главное окно Visual C++, откроем окно «Проводник» и откроем папку c решением Lab1. Заглянем в папку Lab1\_1, созданную мастером приложений для наше­го проекта, и приведём краткое описание размещенных там файлов:

1. Lab1.sln — файл загрузки рабочей области решения.
2. Во вложенной папке Lab1\_1 находится файл Lab1\_1.vcproj — проектный файл, используемый для построения (building) отдель­ного проекта или подпроекта.
3. Debug — папка, в которую будут помещаться файлы, формируемые компилято­ром и сборщиком. Из этих файлов нас будет интересовать, в общем-то, только один — исполняемый файл, имеющий расширение .ехе.

Развернем главное окно Visual C++ с открытой рабочей областью Lab1, чтобы продолжить работу по созданию нашей первой программы. Первое, что бро­сается в глаза — окно Solution Explorer «оживилось», в нем появилось дерево списка файлов, содержащее пиктограммы трех папок: Source Files, Header Files, Resource Files. Так как в консольных приложе­ниях файлы ресурсов не используются, то про последнюю папку сразу забудем. Попробуем заглянуть (щелчком мыши) в первые две папки. Попытка окажется неудачной — папки пусты. Это и неудивительно: ведь мы выбрали в качестве под­типа консольного приложения пустой проект — опцию An empty project. Так что наполнение проекта конкретным содержанием у нас еще впереди.

### Добавление к проекту файлов с исходным кодом

Рассмотрим две ситуации: а) создание ново­го файла, б) добавление существующего файла.

#### 3.3.3.1 Добавление нового файла

В этом случае необходимо выполнить следующие действия:

Выберите в строке меню главного окна команду Project ► Add New Item.... В результате откроется диалоговое окно Add New Item.

1. В окне Categories выберите тип файла (в данном случае Visual C++).
2. В окне Templates выберите C++ File(.cpp)
3. В текстовом поле File Name введите нужное имя файла (в данном случае Lab1\_1);
4. Щелкните на кнопке Add.

После предпринятых шагов можно наблюдать следующие результаты:

- в окне Solution Explorer папка Source Files списка файлов проекта раскроется, и в нее будет помещен файл Lab1\_1.cpp;

- окно Editor засветится мягким белым светом, а в левом верхнем углу его замер­цает текстовый курсор, предлагая ввести какой-нибудь текст.

Введите, например, такой текст:

*#include <iostream> //заголовочный файл для потокового ввода-вывода*

*using namespace std;/\* std – пространство имен, используемое в заголовочном файле\*/*

*int main() //в С++ выполнение программы всегда начинается с функции main()*

*{*

*char str1[80]; // объявление символьного массива str1 размером 80 символов*

*cout << "Welcome to C++ !" <<endl; //Вывод на экран сообщения Welcome to C++ !*

*cout << "Enter the string:" << endl: //Вывод на экран приглашения: Enter the string*

*cin >> str ;*

*/\*Ввод из буфера входного потока одного слова в символьный массив str1. Во время выполнения программы, после вывода на экран приглашения, текст, который Вы хотите ввести, нужно набрать на клавиатуре и нажать клавишу Enter*

*\*/*

*cout << "The value str1= " << endl:*

*cout << strl<< endl:*

*return 0;*

}

В приведенной выше программе для пояснения содержания использованы комментарии (они присутствуют только в исходном файле и исключаются из него при компиляции). В С++ предусмотрены комментарии двух видов:

**//** - однострочный комментарий (от комбинации символов // и до конца строки);

/\* - начало текста комментария, \*/ - конец комментария. Обычно используется для многострочных комментариев.

Программа содержит синтаксические ошибки.

Используя сообщения в окне Output, устраните ошибки в программе.

#### 3.3.3.2 Добавление существующего файла

В этом случае файл с исходным кодом (пусть это будет файл Lab1\_1.cpp) вы уже под­готовили ранее в каком-то текстовом редакторе или скачали по сети. Продолжение должно быть следующим:

Создайте новый проект Lab1\_2.

Скопируйте исходный файл (Lab1\_1.cpp) в папку проекта (в данном случае — Lab1\_2/Lab1\_2).

Выберите в строке меню главного окна команду Project ► Add Existing Item.... В результате откроется диалоговое окно Add Existing Item. В открывшемся диалоговом окне выберите нужный файл (Lab1\_1.cpp) и щелкните на кнопке ОК.

В этом случае вы получите два независимых файла и изменения в одном из них не будут влиять на другой.

Следует иметь в виду, что в файлах проекта может быть только одна функция main(), то есть в проект нельзя включать несколько файлов, которые содержат функцию main().

## Многофайловые проекты

Никаких особых усилий при создании многофайловых проектов вам прилагать не придется: надо просто несколько раз повторить процедуру создания/добавления исходных файлов, описанную выше. Добавляемые исходные файлы обычно представляют собой определения функций или классов, которые вызываются из файла, содержащего функцию main(). В многофайловых проектах обычно присутствуют и заголовочные файлы — они создаются/добавляются после щелчка правой кноп­кой мыши на пиктограмме папки Header Files в окне Solution Explorer; при этом на вкладке Files диалогового окна New выбирается тип файла C/C++ или Header File.

**ПРИМЕЧАНИЕ**

Папки Source Files и Header Files, пиктограммы которых вы видите в окне Solution Explorer, на самом деле физически не существуют, то есть все файлы помещаются в основную папку рабочей области проекта, имя которой было задано при создании проекта в окне *Project Name.* Но согласитесь, что такое упорядочение дерева списка файлов в окне Solution Explorer очень удобно.

## Компиляция, компоновка и выполнение проекта

Эти операции могут быть выполнены или через меню Build главного окна, или с помощью кнопок панели инструментов. Опишем кратко основные команды меню Build:

1. Compile — компиляция выбранного файла. Результаты компиляции выводятся в окно Output.
2. Build — компоновка проекта. Компилируются все файлы, в которых произошли изменения с момента последней компоновки. После компиляции происходит сборка (link) всех объектных модулей, включая библиотечные, в результирующий исполняемый файл. Сообщения об ошибках компоновки выводятся в окно Output. Если обе фазы компоновки завершились без ошибок, то созданный исполняемый файл с расширением .ехе может быть запущен на выполнение.
3. Rebuild All — то же, что и Build, но компилируются все файлы проекта независимо от того, были ли в них произведены изменения.

Если при компоновке многофайлового проекта посредством команды Build вы получаете сообщения об ошибках компиляции или сборки, которые вы не можете объяснить, то рекомендуется воспользоваться командой Rebuild All.

□ Start Debugging (F5)— выполнение исполняемого файла, созданного в результате компоновки проекта в режиме отладки.



Рис. 1. Схема подготовки программы для исполнения в среде компилятора

Откомпилируйте проект, щелкнув на кнопке Build (F7). Диагностические сообще­ния компилятора и компоновщика отображаются в окне вывода Output. Мы надеемся, что у вас все будет в порядке и последняя строка в окне вывода будет выглядеть так:

Lab1\_1.exe - 0 error(s), 0 warning(s)

Теперь запустите приложение на выполнение, щелкнув на зеленом треугольнике (Ctrl+F5).

На приглашение ввести строку введите любую строку, например Hello, World!, и нажмите Enter.

Мы не будем здесь разбираться, почему программа вывела «обрезанную» строку Hello, (проблема ввода-вывода строк будет частично рассмотрена ниже).

Попробуйте заменить в программе оператор вывода

cout << "Welcome to C++ !" << endl;

следующим оператором:

cout << "Добро пожаловать в C++ !" << endl;

Откомпилируйте (точнее говоря, скомпонуйте) программу и запустите ее на вы­полнение. Результат будет настолько безобразным, что рисунок-копию экрана мы здесь приводить не станем.

#### 3.5.1 Конфигурация проекта

Visual C++ позволяет строить проект либо в *отладочной конфигурации* (Win32 Debug), либо в *выпускной конфигурации* (Win32 Release). Мы будем всегда работать с проектами в отладочной конфигурации. Обычно она установле­на по умолчанию. Все же не мешает проверить, с какой конфигурацией на са­мом деле идет работа. Для этого выберите в меню Project пункт Settings... От­кроется диалоговое окно Project Settings. Проверьте, какое значение установлено в окне комбинированного списка Settings For:.. Если это не Win32 Debug, то переключитесь на нужное значение через команду меню Build ► Set Active Con­figuration...

Как закончить работу над проектом

Можно выбрать меню File, пункт Close Solution. А можно просто закрыть прило­жение Visual C++.

#### 3.5.2 Как открыть проект, над которым вы работали ранее

1. Способ первый:

1. Запустите на выполнение Visual C++.
2. Выберите меню File, пункт Open ...
3. В открывшемся диалоговом окне найдите папку с вашим проектом, а в ней — файл ProjectName.vcproj.
4. Откройте этот файл, щелкнув по нему мышью.

2. Способ второй:

■ Запустите на выполнение Visual C++.

1. Выберите меню File, наведите курсор мыши на пункт Recent Workspaces.
2. Если в появившемся меню со списком последних файлов, с которыми шла  
   работа, вы найдете интересующий вас файл ProjectName.vcproj, то щелкните по  
   нему мышью.

3. Способ третий:

1. Не вызывая Visual C++, найдите папку с вашим проектом, а в ней — файл  
   ProjectName.vcproj.
2. Щелкните мышью на файле ProjectName.vcproj.

## Cправочная система

Документация по продуктам Microsoft MSDN доступна, если она установлена вместе с программой на вашем компьютере или если вы подключены к сети Internet.

Если вы нахо­дитесь в окне Editor, то можно пользоваться интерактивной справкой: поставьте текстовый курсор на интересующий вас оператор или библиотечную функцию C++ и нажмите клавишу F1. Тотчас будет вызвана справочная система MSDN с предоставлением необходимой информации. Если запрошенный термин встречается в разных разделах MSDN, то сначала появится список разделов. В списке разделов выберите тот, в котором упоминается «Visual Studio». Выберите нужную версию Visual Studio и в ней раскрывающиеся списки: /Справочник /Справочники по языкам /Справочник по языку С++ /Операторы /Операторы итерации/ Оператор for.

***Задание.***

Найдите в сети Internet на сайте ***docs.microsoft.com*** информацию об операторе ***for.*** Если она написана на английском языке, это не должно вас останавливать. Для программирования в среде Visual C++ Вам следует изучить английский на столько, чтобы понимать сообщения компилятора и тексты справочной системы! Это не так уж сложно, так как число сообщений ограничено, а в справочной системе широко использованы копирование и вставка.

## Проблемы с вводом-выводом кириллицы

Работа в среде Visual C++ (в режиме консольных приложений) сопряжена с определенными неудобствами, вызванными различными стандартами кодировки символов кириллицы в операционных системах MS DOS и Windows. Напомним, что весь ввод-вывод в консольном окне идет в кодировке стандарта ASCII, а текст в исходных файлах, набираемый в текстовом редакторе Visual C++, имеет кодировку в стан­дарте ANSI. Символы кириллицы, в отличии от символов латинского алфавита, в этих стандартах имеют различную кодировку.

Для устранения неудобств работы с кириллицей вставьте в программу в начало функции main() оператор

system(" chcp 1251 > nul");

Это позволит вам при выполнении лабораторных работ (но не во всех случаях) устранить проблемы, связанные с использованием кириллицы. Если проблема не устранилась, то установите в окне консоли, связанном с программой, по умолчанию шрифт *Lucida Consol* . Для этого нужно открыть меню свойств консоли, выбрать пункты *умолчания, шрифт* и выполнить необходимые установки.

*////////////////////////////////////////////*

#include <iostream >

using namespace std;

int main(){

system(" chcp 1251 > nul");

char str1[80];

cout << "Welcome to C++ !" << endl;

cout << "Введите строку:" << endl; //введите строку: Привет, мир!

cin >> str1;

cout << "Знaчение строки str1 = " << endl;

cout << str1 << endl;   
system("pause");

return 0;

}

//////////////////////////////////////////////////////////

Ничего не принимайте на веру. Откомпилируйте эту программу и убедитесь, что она дает ожидаемый результат. Для того, чтобы можно было просмотреть результаты работы программы в окне DOS до его закрытия, удобно в конце программы перед оператором return вставить оператор system("pause").

## Замечания по потоковому вводу-выводу

***Ввод из буфера входного потока одного слова в символьный массив str1.*** Предварительно весь текст нужно набрать на клавиатуре и нажать клавишу Enter, после чего ***весь текст*** поступит в буфер входного потока *cin*, а оттуда его можно ***пословно*** считывать с помощью перегруженной для входного потока операции >>(поразрядный сдвиг вправо) в переменную, стоящую справа от >>, в данном случае это символьный массив str1. ***Одна операция считывает одно слово***. Если повторить операцию, то из буфера входного потока в массив str1 (в его начало, поверх считанного ранее слова) будет записано следующее слово. Пробел является по умолчанию разделителем между словами и на его место операция >> вставляет символ конца строки (его код равен 0 и он не печатается). Если буфер входного потока пустой, то при выполнении операции >> на экран выведется запрос на ввод данных в виде мерцающего курсора. Программа остановится и будет ожидать ввода данных с клавиатуры. Ввод комбинации из двух одновременно нажатых клавиш Ctrl+z (печатается как ^Z) является символом «Конец файла» при вводе с клавиатуры.

После нажатия клавиши ***Enter*** введенный текст поступит в буфер входного потока, откуда его можно будет пословно, с помощью операции cin >> str1, считывать в массив символов str1, определенный в программе. Только после этого введенные данные становятся доступными для обработки.

Потоковый ввод-вывод ***автоматически выполняет форматирование данных в соответствии с типом переменной***, стоящей справа от знака операции. Это означает, что если бы мы ввели с клавиатуры символы 123 и в программе использовали бы операторы

int n; cin >> n; ,  
то целочисленной переменной ***n*** было бы присвоено значение 123. Но если бы мы ввели символы a23, то программа обнаружила бы ошибку формата и не выполнила бы операцию cin >> n;.

Потоковый ввод-вывод подробно будет рассматриваться при изучении объектно-ориентированного программирования, а пока вам нужно запомнить синтаксис операторов потокового ввода–вывода, чтобы использовать его в разрабатываемых программах.

***Задание.***

1. Разработайте программу, которая вводит два целых числа и выводит их сумму.

2. С целью освоения потокового ввода-вывода измените приведенную выше программу так, чтобы она печатала введенную строку *Привет, мир*! полностью, используя для хранения введенного текста только один буфер (массив символов str1). В остальном вид экрана должен остаться прежним. Строка вводится с клавиатуры только один раз.

## Работа с отладчиком

Полное описание возможностей встроенного отладчика Visual C++ и приемов ра­боты с ним может потребовать отдельной книги, настолько объемна эта тема. По­этому здесь приведены только начальные сведения о работе с отладчиком Visual C++. Проще всего это сделать, написав программу, заведомо содержащую несколько ошибок, а затем показав, как с помощью отладчика можно найти и исправить эти ошибки.

В частности, мы научимся устанавливать в программе *точки прерывания* и выпол­нять ее до заданной точки. Когда во время выполнения встречается точка преры­вания, программа останавливается, а на экране появляется отлаживаемый код. Это дает возможность детально выяснить, что происходит в программе.

Кроме того, программу можно выполнять последовательно, строку за строкой — такой процесс называется *пошаговым выполнением.* Этот режим позволяет следить за тем, как изменяются значения различных переменных. Иногда он помогает по­нять, в чем заключается проблема: если обнаруживается, что переменная прини­мает неожиданное значение, то это может послужить отправной точкой для выяв­ления ошибки. После обнаружения ошибки ее можно исправить и выполнить программу заново в отладочном режиме.

Назовем нашу программу (проект) именем Buggy. Программа должна вычислять среднее арифметическое первых пяти натуральных чисел: 1-5. Нетрудно догадаться, что ответ должен быть равен 3, однако из-за специально сделанных ошибок программа первоначально будет выдавать неправильный ответ.

Для создания программы выполните следующие действия:

1. Создайте проект типа «консольное приложение» с именем Buggy.
2. Добавьте к проекту файл buggy.срр и заполните его следующим текстом:

#include <iostream>

using namespace std;

int main()

{ const N = 5;

int a[N] = {1, 2, 3, 4, 5};

float sum, average;

int i;

for (i = 1: i < N: i++)

sum = sum+a[i];

average = sum / N;

cout << "average = " << average << endl;

return 0;

}

□ Откомпилируйте проект.

□ Запустите программу на выполнение.

Вы должны увидеть в консольном окне приложения нечто вроде следующего ре­зультата:

average = -2.14748e+007,

т. е. программа вычислила, что среднее арифметическое первых пяти целых чисел  
равно -21474800 (на вашем компьютере может быть и другое число), а это мало  
похоже на число 3.0.

Начнем отладку нашей злополучной программы.

### Установка **точки** прерывания

Точка прерывания позволяет остановить выполнение программы перед любой *выполняемой* инструкцией (оператором) с тем, чтобы продолжать выполнение про­граммы либо в пошаговом режиме, либо в непрерывном режиме до следующей точ­ки прерывания.

Чтобы задать точку прерывания перед некоторым оператором, необходимо уста­новить перед ним текстовый курсор и нажать клавишу F9. Точка пре­рывания обозначается в виде коричневого кружка на левом поле окна редактиро­вания. Повторный щелчок на кружке снимает точку прерывания. В про­грамме может быть несколько точек прерывания.

### Выполнение программы до точки прерывания

Программа запускается в отладочном режиме с помощью команды Build ► Start Debug (или нажатием клавиши F5).

В результате код программы выполняется до строки, на которой установлена точ­ка прерывания. Затем программа останавливается и отображает в окне Editor ту часть кода, где находится точка прерывания, причем желтая стрелка на левом поле указывает на строку, которая будет выполняться на следующем шаге отладки. Продолжим демонстрацию описываемых средств на примере программы Buggy:

1. Установите точку прерывания перед оператором for.
2. Запустите программу в отладочном режиме, нажав клавишу F5.

Обратите внимание — в Visual C++ на панели инструментов появилось меню Debug. Загля­нем в него из любопытства.

Среди различных команд этого меню особый интерес представляют команды Step Into (F11), Step Over (F10), Step Out (Shift+F11), Run To Cursor (Ctrl+F10) и Stop Debug­ging (Shift+F5).

Выбор последней команды (или нажатие комбинации клавиш Shift+F5) вызывает завершение работы с отладчиком.

### Пошаговое выполнение программы

Нажимая клавишу *F10*, можно выполнять один оператор программы за другим.

Предположим, что при пошаговом выполнении программы вы дошли до строки, в которой вызывается некоторая функция func1(). Если вы хотите пройти по коду вызываемой функции, то надо нажать клавишу *F11*. Если же внутренняя рабо­та функции вас не интересует, а интересен только результат ее выполнения, то надо нажать клавишу *F10*.

Допустим, что вы вошли в код функции func1(), нажав клавишу *F11*, но через не­сколько строк решили выйти из него, т. е. продолжить отладку после возврата из функции. В этом случае надо нажать клавиши *Shift+F11*.

Существует и другая возможность пропустить пошаговое выполнение некоторого куска программы: установите текстовый курсор в нужное место программы и на­жмите клавиши *Ctrl+F10*.

Продолжим отладку программы Buggy:

□ Нажмите клавишу F10.

Указатель следующей выполняемой команды переместится на оператор

***sum =sum + a[i];***

#### 6.3.1 Проверка значений переменных во время выполнения программы

Чтобы узнать значение переменной ***sum***, в которой будет накапливаться сумма эле­ментов массива ***а***, задержите над ней указатель мыши. Рядом с именем перемен­ной на экране появляется подсказка со значением этой переменной:

***sum*** = -1.07374e+008 (или с другим произвольным значением).

Ага!!! Ведь еще не было никакого суммирования и, следовательно, переменная ***sum***, по идее, должна быть равна нулю. Вот где собака зарыта! Мы забыли обнулить переменную ***sum*** до входа в цикл.

□ Нажмите комбинацию клавиш *Shift+F5*, чтобы выйти из отладчика и исправить  
найденную ошибку. Найдите строку с объявлением

***float sum, average;***

и добавьте в нее инициализацию переменной sum:

***float sum = 0, average;***

□ Откомпилируйте заново проект — кнопка *Build (F7).*

□ Запустите на выполнение — кнопка *Execute Program (Ctrl+F5).*  
Вы получите новый результат:

***average = 2.8***

Это как бы теплее, но все равно еще неправильно. Нажмите любую клавишу для завершения работы приложения. Необходимо продолжить отладку:

1. Установите точку прерывания перед оператором ***for***. .
2. Запустите программу в отладочном режиме, нажав клавишу *F5*.
3. Нажмите клавишу *F10*.

Указатель следующей выполняемой команды переместится на оператор

***sum = sum+a[i];***

□ Нажмите клавишу *F10*.

Указатель следующей выполняемой команды переместится на оператор

***for (i = 1; i < N; i++)***

□ Задержите теперь над переменной ***sum*** указатель мыши. Рядом с именем переменной на экране появится подсказка со значением этой переменной

sum = 2.

Стоп!!! Позади 1-я итерация цикла, и в переменной ***sum*** должно находиться значе­ние первого элемента массива ***а***, т. е. число 1. А мы имеем число 2, то есть значение второго элемента массива ***а***. Пришло время вспомнить, что в языке C++ нумера­ция элементов массива начинается с 0! Поэтому ошибка находится в заголовке цикла:

***for (i = 1; i < N;i++)***

и ее нужно немедленно исправить. Повторите действия, направленные на исправ­ление ошибки:

1. выйти из отладчика,
2. исправить текст программы в операторе for:

***for (i = 0: i < N; i++)***

1. откомпилировать,
2. запустить на выполнение.

Если в процессе исправления вы не внесли новых ошибок, то должен получиться результат:

average = 3.0.

Итак, все ОК! — программа работает правильно.

В заключение отметим, что отладчик предоставляет и другие возможности для наблюдения за значениями переменных во время выполнения программы.

#### 6.3.2 Окна Auto, Local и Watch

Помимо экранной подсказки, переменная sum со своим значением отображается в окнах **Auto** и **Local** расположенных в левом нижнем углу экрана (программист может управлять вызовом и расположением этих окон). В этом окне приведены значения последних переменных, с которыми работал Visual C++.

Кроме этого, в окне Watch 1, которое находится в правом нижнем углу, можно за­дать имя любой переменной, за значениями которой вы хотите понаблюдать.

Более подробно о работе с этими окнами можно прочитать в справочной системе через меню **Help** главного окна Visual C++.

***Задание.***

Выполните программу п. 6 по шагам, используя для просмотра значений переменных окна Local и Watch1.

## 3.10 Содержание отчета по лабораторной работе

По данной работе отчет не представляется. Результаты выполнения пунктов задания показываются преподавателю в процессе выполнения ЛР.

## 3.11 Контрольные вопросы

1. Как создать в VS проект типа консольного приложения на языке С++?
2. Как добавить в проект новый файл с кодом программы?
3. Как в VS выполнить компиляцию, компоновку и выполнение проекта?
4. Как закончить работу над проектом?
5. Как открыть проект, над которым вы работали ранее?
6. Как вывести на экран результаты выполнения программы?
7. Как выполнять программу по одной строке за раз?
8. Как следить за значениями переменных во время выполнения приложения?
9. Как автоматически останавливать выполнение программы в определенных точках кода?

7. Как задавать условия для останова?

## 3.12 Рекомендуемые источники информации

1. Справочная система Microsoft MSDN https://docs.microsoft.com/

# Лабораторная работа 2. Программирование разветвляющихся алгоритмов

## Цель работы

Целью работы является приобретение студентами следующих навыков:

- использование метода структурного программирования (пошаговой детализации) при разработке программ;

* использование логических операций для составления «сложных» условий;
* использование операторов ***if, if-else*** и ***switch*** при программировании разветвляющихся алгоритмов;
* изображение операторов ***if, if-else*** и ***switch*** на схемах алгоритмов.

Планируемое время выполнения работы- 6 часов.

## Задание

Вывести на экран показания часов.

Исходные данные – часы и минуты в формате целых чисел. Результат должен быть представлен в смешанной текстово-числовой форме с учетом падежных окончаний в следующем виде:

2 48 – 2 часа 48 минут ночи,

5 00 – 5 часов утра ровно,

12 00 – полдень,

12 23 – 12 часов 23 минуты дня,

13 12 – 1 час 13 минут дня,

21 33 – 9 часов 33 минуты вечера,

00 00 – полночь,

03 01 – 3 часа 1 минута ночи.

Использовать следующие границы частей суток:

5 час. <= утро < 12 час.,

12 <= день < 18 час.,

18 <= вечер <= 23 час.,

0 час. <= ночь < 5 час.

В случае ввода невозможных показаний часов выдать соответствующее сообщение, например:

24 03 – введены недопустимые данные.

Программа должна быть разработана в соответствии с приведенной в методических указаниях укрупненной схемой алгоритма.

По результатам работы представить отчет.

## Рекомендации по разработке программы

Создание программы - процесс сложный, поэтому практически с любо­го этапа возможен возврат на предыдущие этапы для исправления ошибок или принятия других проектных решений. Чаще всего такого рода возвраты являются следствием ошибок, допущенных при разработке алгоритма.

Для уменьшения ошибок в алгоритмах программ при их разработке следует использовать метод пошаго­вой детализации.

При использовании данного метода разработку алго­ритмов выполняют поэтапно. На первом этапе описывают решение постав­ленной задачи, выделяя подзадачи и считая их решенными. На следующем - аналогично описывают решение подзадач, формулируя уже подзадачи следу­ющего уровня. Процесс продолжают до тех пор, пока не дойдут до подзадач, алгоритмы решения которых очевидны. При этом, описывая решение каждой задачи, желательно использовать не более одной-двух конструкций, таких как цикл или ветвление, чтобы четче представлять структуру программы. Так как для большинства студентов это первая программа на С++, то для её разработки достаточно использовать целочисленные переменные, текстовые константы и операторы ветвления ***if, if-else*** и ***switch***, а для ввода и вывода данных – потоки *cin* и *cout*, с которыми вы познакомились в первой лабораторной работе.

Шаг1. Определяем общую структуру программы.

Разбиение сложного алгоритма на подзадачи (сегменты программы) следует выполнять таким образом, чтобы каждая подзадача имела физический смысл и правильность ее выполнения можно было бы проверить, не выполняя последующие действия или выполняя их небольшую часть, например, печать результатов разработанных подзадач. Помните, что алгоритм – это последовательность действий и одно из основных его свойств – дискретность. Это означает, что в линейных (не параллельных) алгоритмах каждое последующее действие должно выполняться после завершения предыдущего.

Перед тем, как рисовать схему алгоритма, нужно описать используемые в алгоритме переменные, указав их назначение и тип, например:

int h – значение часов;

int m– значение минут.

На рисунке 1 показан вариант укрупненной схемы алгоритма программы.   
При выполнении задания лабораторной работы ***необходимо*** реализовать именно этот алгоритм.

После того, как схема алгоритма нарисована, не выполняя кодирования, все его ветви нужно проверить с помощью разработанных для этих целей контрольных примеров.

* Шаг 2. Детализируем операцию ввода и контроля исходных данных. После завершения разработки детального алгоритма первой подзадачи ее следует закодировать, заменяя, если необходимо, неразработанные подзадачи «заглушками» (заглушка не выполняет никаких действий, а только печатает свое имя или возвращает тестовые значения переменных), выполнить и протестировать. Например, для ввода и контроля исходных данных остальные действия не требуются и для отладки этого блока достаточно выводить на печать введенные данные.

***Не следует сразу кодировать всю программу, так как в этом случае значительно сложнее обнаруживать и исправлять ошибки***. Пишите и отлаживайте программу по частям, так, чтобы на каждом шаге разработки программа могла выполняться (была бы «живой»). Это позволит обнаружить и устранить возможные ошибки на ранней стадии их возникновения и существенно сократить общее время разработки программы.

Шаг 3. Детализируем подзадачу определения времени суток. Ее удобнее выполнить до обработки часов, так как при выводе результатов значения часов и их падежные окончания могут измениться. Например, вместо 15 часов мы должны печатать 3 часа дня. Это можно учесть, если запомнить время суток в переменной timeOfDay типа int, например (1 - утро, 2-день, 3- вечер, 4-ночь), и выполнить операцию h=h-12, если h>12.

Шаги 4, 5. Детализируем операции обработки часов и минут. Результаты обработки запоминаются в переменных типа int, например,

hPadeg - падежное окончание для вывода числа часов,

mPadeg - падежное окончание для вывода числа минут.

Возможные значения hPadeg равны:

1, если надо печатать «час»;

2, если надо печатать «часа»;

3, если надо печатать «часов».

Для определения падежных окончаний слов при печати значений минут эффективно использовать операцию деления по модулю (эта операция определена для целых чисел и в С++ обозначается знаком **%**), а для выделения диапазона чисел с одинаковыми значениями падежных окончаний использовать операторы больше, меньше, больше равно, меньше равно (>, <, >=, <=).

Например, если ( m>4 && m<21 ), то надо печатать ***минут,*** а иначе,

если ( m%10 == 1), то есть для m=21, 31, 41 и 51 надо печатать ***минута***.



Рис.1. Укрупненная схема алгоритма

Шаг 6. При выводе результатов значения переменных hPadeg и mPadeg будем использовать в операторе switch в качестве ключа при печати результата с помощью операторов потокового вывода (формирование строки на экране монитора операторами потокового вывода рассматривалось в лабораторной работе 1). Вид строк с результатами выполнения программы должен точно соответствовать строкам, указанным в задании. Например:

2 48 – 2 часа 48 минут ночи

5 00 – 5 часов утра ровно

12 00 – полдень.

Таким образом, при использовании метода пошаговой детализации ***на каждом шаге решается одна достаточно простая задача, что существенно облегчает разработку программы и является основным достоинством метода пошаговой детализации.***

Например, на первом шаге мы вводим исходные данные (время в виде двух целых чисел: часы и минуты), проверяем введенные данные, чтобы исключить ввод недопустимых значений, распечатываем введенные данные и выполняем тестирование, то есть проверку ***всех*** возможных вариантов ввода. На этапе отладки программы всегда следует распечатывать введенные данные, так как неверный ввод – типичный источник ошибок. После успешного тестирования отладочную печать нужно закомментировать (не удалить), ибо согласно одному из законов Мерфи «последняя ошибка в программе на самом деле предпоследняя» и как только вы удалите отладочную печать она Вам тут же потребуется.

В программировании задачу обычно можно решить несколькими способами. Какой из возможных алгоритмов следует использовать? Основными характеристиками алгоритмов являются сложность (понятность), время выполнения, занимаемая оперативная память, время разработки. Для выполнения заданий лабораторных работ (и не только) следует использовать наиболее понятный (простой) алгоритм. Такие алгоритмы (если не предъявляются специальные требования к времени выполнения или занимаемой оперативной памяти) обычно имеют лучшие обобщенные характеристики и меньшее время разработки.

Использование метода пошаговой детализации способствует разработке понятных алгоритмов и сокращает время разработки программы, делая его прогнозируемым.

## Требования к отчету

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

- Постановка задачи;

- Разработка алгоритма;

- Кодирование (соответствующий раздел отчета называется «Текст программы»);

- Тестирование (соответствующий раздел отчета называется «Анализ результатов»).

В разделе «Постановка задачи» должен быть приведен текст задания и согласованные с преподавателем (заказчиком) уточнения, если они требуются (например, как выводить время 00:03 - 0 часов 3 минуты ночи или 12 часов 3 минуты ночи, или существует ли время 24:00).

Раздел «Разработка алгоритма» должен содержать следующую информацию:

* краткое описание (обоснование) алгоритма,
* описание входных, выходных и вспомогательных данных с указанием их идентификаторов и типов,
* схему алгоритма, состоящую из двух частей: общей (укрупненной) схемы и уточненной схемы одного из блоков:
  + Ввод и контроль исходных данных;
  + Обработка часов;
  + Обработка минут;
  + Обработка времени суток.

(Номер блока равен остатку от деления порядкового номера студента в списке группы на 4 плюс 1.)

В разделе «Текст программы» должен быть приведен листинг программы, включающий необходимые комментарии.

В разделе «Анализ результатов» должны быть приведены результаты проверки работы всех ветвей алгоритма. Результаты следует оформить в виде таблицы, а для двух наборов исходных данных должны быть приведены распечатки экранов. Для экономии краски в принтере изображения экранов должны иметь белый фон.

Отчет должен быть распечатан на принтере на листах бумаги формата А4, скрепленных в левом верхнем углу с помощью степлера, и подписан исполнителем с указанием даты сдачи отчета преподавателю.

На последней странице методических указаний к данной лабораторной работе показан титульный лист, который должен использоваться в отчетах по всем лабораторным работам курса (изменяться должны только номер, название лабораторной работы и фамилии студента и преподавателя).

## Контрольные вопросы

1. Из каких компонентов состоит программа на С++?

2. Дайте определение понятия «алгоритм».

3. Приведите основные свойства алгоритма выполнения программы.

4. Что такое «метод пошаговой разработки программы»?

5. По каким правилам производится разделение алгоритма на структурные блоки?

6. Приведите основные достоинства метода пошаговой детализации и объясните, за счет чего они достигаются.

7. Приведите фрагмент программы с использованием оператора ***if*** и представление этого фрагмента на схеме алгоритма.

7. Приведите фрагмент программы с использованием оператора ***if-else*** и представление этого фрагмента на схеме алгоритма.

7. Приведите фрагмент программы с использованием оператора ***switch*** и представление этого фрагмента на схеме алгоритма.

## Рекомендуемые источники информации

* 1. Р. Лафоре. Объектно-ориентированное программирование в С++   
     (главы 1, 2, 3). Издательство ПИТЕР, 2004
  2. Ван Тассел Д . Стиль, разработка, эффективность, отладка и испытание программ (общие вопросы разработки программ), «Мир», 1981
  3. <http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.
  4. Г.С. Иванова. Основы программирования, (метод пошаговой детализации). Высшая школа. 2005
  5. Т.А. Павловская, Ю.А. Щупак. С/С++ Структурное программирование, (примеры разработки программ с комментариями). Практикум. ПИТЕР, 2003

**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №2

«Программирование разветвляющихся алгоритмов»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-13 |  | преподаватель каф. ИУ5 |
| Иванов Иван |  | Петров П.И. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2019 г.

# Лабораторная работа 3. Табулирование функций с использованием рядов Тейлора

## Цель работы

Целью работы является приобретение студентами следующих навыков:

* использование операторов ***for, while и do-while*** при программировании циклических алгоритмов;
* использование вложенных циклов;
* преобразование исходных выражений с целью получения эффективных (с точки зрения точности получаемых результатов и времени выполнения) расчетных соотношений при выполнении расчетов по формулам;
* изображение циклов ***for, while и do-while*** на схемах алгоритмов;
* управление точностью вычислений и точностью представления на мониторе результатов вычислений;
* использование манипуляторов ***fixed, setw*** и ***setprecision*** для форматирования потокового вывода
* тестирование программы и исправление ошибок в режиме отладки.

Планируемое время выполнения работы- 6 часов.

## Задание

Номер варианта задания равен остатку от деления порядкового номера студента в списке группы на 13. Если остаток равен 0, то вариант равен 13.

Каждое задание содержит четыре задачи.

Для удобства выполнения ЛР и демонстрации результатов преподавателю лабораторная работа должна быть оформлена в Visual Studio как одно решение, состоящее из 4-х проектов, в каждом из которых решается одна задача.

Варианты задания приведены в Приложении 1.

## Рекомендации по выполнению работы

### Указание к задаче 1 задания

Для повторения или завершения выполнения программы используйте цикл   
do-while, который должен включать в себя запрос “Продолжить работу? (y/n)” и ввод с клавиатуры соответствующего символа. Это позволит запускать программу с новыми данными, не завершая ее. Используйте такой прием в последующих ЛР в тех случаях, когда требуется многократно запускать программу с различными исходными данными (например, для отладки или демонстрации работы преподавателю).

### Указание к задаче 2 задания

Объясните результат: при *а = 5,7* получили *S =* 147 450,   
а должно быть 147450.3 (***a*** и ***S*** – действительные числа типа ***float***).   
Обеспечьте нужную точность представления результата, используя манипуляторы ***setw(n)***  и ***setprecision(n)*** для потокового вывода.

Объясните результат при *а = 5,125* и при *а = 5,135*. Обеспечьте нужную точность *вычисления* и *представления* результата для этого случая.

### Указание к задаче 3 задания

При вычислении значения очередного члена ряда используйте значение предыдущего члена, для чего следует вручную получить соотношение вида

k(x,n) = Ai (x,n) / Ai-1(x,n i-1);

Вычисление членов ряда, начиная со второго (а может и с третьего), следует выполнять по рекурсивной формуле:

Ai = Ai-1\* k;

Это упростит вычисления, повысит их точность и позволит избежать возможного переполнения при вычислении факториалов и степеней.

Заданная точность обеспечивается суммированием членов ряда вплоть до слагаемого, абсолютное значение которого меньше заданной погрешности. Начальное значение погрешности 0.000001. Погрешность ***вводится с клавиатуры и может изменяться***.

Для представления результата в виде таблицы при формировании строк используйте манипуляторы ***setw*** и ***setprecision***. При выводе таблицы точность представления данных должна быть не ниже требуемой точности их вычисления.

Близость значений суммы членов ряда Тейлора - *S(x)* и значений, вычисленных по формуле с помощью библиотечных функций -*Y(x)* (отличие должно быть меньше *eps* во всем диапазоне значений *х*)*,* указывает на правильность их вычисления.

Выполните задание для погрешностей eps=0.000001 и eps=0.00000001.

### Указание к задаче 4 задания

Если требуется вычислить результат при ***n,*** меньшем, чем 3, 5 или10 , то значения последующих промежуточных сумм на экран не выводить.

Значения математических констант, таких как ***pi, e***, хранятся в файле ***math.h.*** Так как их значения не определяются в стандарте языка С++, то при простом включении файла в программу эти константы недоступны. Для доступа к ним необходимо перед оператором препроцессора #include <math.h> выполнить оператор #define \_USE\_MATH\_DEFINES . Имена констант найдите в файле math.h.

## Содержание отчета.

Номер варианта задания должен быть указан на титульном листе после наименования работы.

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

* Постановка задачи;
* Разработка алгоритма;
* Кодирование (соответствующий раздел отчета называется «Текст программы»);

- Тестирование (соответствующий раздел отчета называется «Анализ результатов»).

Результаты выполнения четырех задач задания оформляются в одном отчете.

В разделе «Постановка задачи» должен быть приведен текст задачи и согласованные с преподавателем уточнения, если они требуются.

В разделе «Разработка алгоритма» должно быть приведено:

* описание используемых переменных с указанием наименования, типа (int, float, и т.п.) и назначения в программе,
* определение расчетного соотношения для вычисления членов ряда (для задач 3 и 4) и блок-схема алгоритма (только для третьей задачи).

Раздел «Кодирование» должен содержать листинг программы с необходимыми комментариями.

В разделе «Тестирование» должны быть приведены результаты выполнения задания. Для третьей задачи результаты следует оформить в виде таблицы.

Для экономии краски при печати, изображения экранов должны иметь ***белый фон***.

Отчет должен быть распечатан на принтере на листах бумаги формата А4, скрепленных в левом верхнем углу с помощью степлера, и подписан исполнителем с указанием даты сдачи отчета преподавателю. Страницы отчета должны быть пронумерованы.

## Контрольные вопросы

1. Библиотечная функция exit() предназначена для выхода из:

а) цикла, в котором она содержится;

б) функции, в которой она содержится;

г) программы, в которой она содержится.

2. Оператор break производит выход:

а) только из цикла наибольшей глубины вложенности;

б) только из ветвления switch наибольшей глубины вложенности;

в) из всех вложенных циклов и ветвлений;

г) из цикла или ветвления наибольшей глубины вложенности.

3. В какое место программы передается управление после выполнения  
 оператора continue внутри цикла?

4. Назовите основное назначение каждого из трех выражений, входящих в состав оператора цикла for.

5. Для чего и как используются манипуляторы ***fixed, setw*** и ***setprecision* ?**

6. Как можно повысить точность вычислений при программировании расчётов по формулам?

7. В чём отличие понятий точности вычислений и точности представления результатов на экране монитора?

8. Определите в п. 3.3 задания в режиме отладки сумму первых трех членов ряда Тейлора

и убедитесь в её правильности.

## Рекомендуемые источники информации

1. Р. Лафоре. Объектно-ориентированное программирование в С++. Издательство ПИТЕР, 2004 г. гл.3.

2. Н. Вирт. Алгоритмы + Структуры данных = Программы, «Мир», 1985 г.

3. Г.С. Иванова. Основы программирования. Высшая школа. 2005 г.

4. <http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.

## Варианты задания

#### 7.1 Вариант 1

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m вводите с клавиатуры.
2. Составьте программу для вычисления:



Значение *а* вводите с клавиатуры.

3. Разработайте программу, которая вычисляла бы значение суммы 

и значение функции *Y(x)* = sin*(* *х) ,* где 0≤ *х* ≤1, с шагом *h=0.2.*

Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x)* и *N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:



Натуральное значение n введите с клавиатуры. Значение *х* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.2 Вариант 2

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.

2. Составьте программу для вычисления:



Значение *a* введите с клавиатуры.

3. Составьте программу вычисления значения суммы 

и функции в диапазоне от 0 до 1 с шагом *h=0.2.*

Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле: .

Натуральное значение *n* введите с клавиатуры. Значения *x* и *а* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.3 Вариант 3

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.

2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы 

и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:  
 для |x|≤1

Натуральное значение *n* введите с клавиатуры. Значения *x* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.4 Вариант 4

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.

2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы  и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:  


Натуральное значение *n* введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных результатов при *n* равном 3, 5 и 10

#### 7.5 Вариант 5

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.
2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы 

и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:  


Натуральное значение *n* введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных результатов при *n* равном 3, 5 и 10.

#### 7.6 Вариант 6

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры

2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы 

и функции в диапазоне от 0 до 1 с шагом *h=0.2* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле: .

Натуральное значение *n* введите с клавиатуры. Значения *x* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.7 Вариант 7

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры
2. Составьте программу для вычисления:



Значение *a* введите с клавиатуры.

3. Составьте программу вычисления значения суммы  и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:  
.

Натуральное значение *n* введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных результатов при *n* равном 3, 5 и 10.

#### 7.8 Вариант 8

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.
2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы 

и функции в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x)*

*и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле: .

Натуральное значение *n* введите с клавиатуры. Значения *x* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.9 Вариант 9

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры
2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы  и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле: .

Натуральное значение *n* введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных результатов при *n* равном 3, 5 и 10.

#### 7.10 Вариант 10

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.

2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы 

и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле: .

Натуральное значение *n* введите с клавиатуры. Значения *x* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.11 Вариант 11

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры.
2. Составьте программу для вычисления:



Значение *a* введите с клавиатуры.

3. Составьте программу вычисления значения суммы  и функции  в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *y* по формуле:

.

Натуральное значение *n* введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных результатов при *n* равном 3, 5 и 10.

#### 7.12 Вариант 12

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры
2. Составьте программу для вычисления:



Значение *a* введите с клавиатуры

3. Составьте программу вычисления значения суммы  и функции *Y(x)* = *cos(2х)-1* в диапазоне от 0 до 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:



Натуральное значение n введите с клавиатуры. Значение *х* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

#### 7.13 Вариант 13

1. Из первых *n* натуральных чисел найдите сумму тех из них, которые делятся на 5 и не делятся на m (m<n). Натуральные значения *n* и m введите с клавиатуры
2. Составьте программу для вычисления:



Значение *а* введите с клавиатуры.

3. Составьте программу вычисления значения суммы  и функции  в диапазоне 0≤ *x<* 1 с шагом *h=0.2.* Вычисление суммы ряда Тейлора производите с погрешностью, не превышающей 0.000001. Результат представить в виде таблицы (без рамок), которая содержит четыре столбца со значениями *x*, *Y(x), S(x) и N,* где *N* - номер последнего слагаемого.

4. Напишите программу для вычисления *у* по формуле:



Натуральное значение n введите с клавиатуры. Значение *х* также введите с клавиатуры. Обеспечьте возможность, не завершая программу, вычислить *y* для нескольких значений *n* и выведите на экран значения промежуточных (частичных) сумм при количестве слагаемых 3, 5 и10.

# Лабораторная работа 4 Численные методы решения нелинейных уравнений

## Цель работы.

- Программирование численных методов решения нелинейных уравнений.

- Сравнительный анализ методов простой итерации, половинного деления и метода Ньютона.

- Использование функций и указателей на функцию.

Планируемое время выполнения работы - 4 часа.

## Задание.

1. Разработайте программу для вычисления корней уравнения

x – k\*cos(x) = 0

при k=1 простой итерацией, половинным делением и методом Ньютона с погрешностью eps < 0.000001 и eps < 0.00000001. Для каждого из трех методов определить (и вывести на экран) количество шагов алгоритма, использованных для получения результата.

2. Выполнить п.1 для k=5 и объяснить результаты.   
Программа должна обеспечивать возможность многократного выполнения пунктов 1 и 2 для различных значений параметров:

- начального приближения *x0*,

- коэффициента *k,*

- требуемой точности вычислений *eps*.  
Для реализации этого требования используйте функцию «Меню».

3. Разработайте «универсальную» функцию для вычисления корней уравнений вида *y(x)=0* методом половинного деления, используя в качестве ***параметра указатель на функцию***.

Примените эту функцию для нахождения всех корней уравнения *y=x-5\*cos(x)* и   
*y=x-10\*cos(x),*  добавив следующие параметры:

eps - требуемая точность вычислений

xl и xr - левая и правая границы интервала, содержащего ровно один корень.

## Рекомендации по выполнению работы

Численному решению уравнения

f(x) = 0 (1)

должно предшествовать хотя бы грубое исследование вопросов существования и положения корней.

***Итерационные методы***

Заданное уравнение f(x) = 0 приводят к виду

x = ϕ(x). (2)

Выбирая некоторое начальное приближение Х0, вычисляют последовательные приближения

Хj+1 = ϕ(Xj), (j=0, 1, 2, …).

Сходимость таких приближений к искомому решению Х требует отдельного исследования. Сходимость зависит прежде всего от вида функции, а также от начального приближения. В данной лабораторной работе такие исследования не делаются, но в пункте 2 задания решения методом Ньютона и методом простой итерации могут расходиться. Для того, чтобы программа нахождения корней этими методами не зацикливалась, при использовании этих методов следует ограничивать максимальное число итераций Nmax, например, Nmax < 100.

Возможны различные способы приведения уравнения (1) к виду (2).

***Простая итерация***

Хj+1 = Xj - f(Xj).

***Метод Ньютона (метод касательной)***

Хj+1 = Xj - f(Xj)/ f ′(Xj).

***Метод половинного деления***

Этот метод является наиболее подходящим для вычисления корней уравнений вида *f(x)=0* с помощью компьютера.

Для использования этого метода нужно задать границы интервала [xl, xr] на оси абсцисс, содержащего *ровно один корень*, и требуемую точность вычислений.

Суть метода заключается в следующем. Выбирают Х на середине интервала [xl, xr] и определяют f(X). Если |f(X)| < eps, то середина интервала считается корнем уравнения, иначе корень ищется на том интервале из двух полученных, для которого значения функции на концах имеют разные знаки. Действия повторяются до тех пор, пока интервал >eps. Перед входом в цикл необходимо проверить, не являются ли границы интервала корнями уравнения f(X)=0.

Пусть

* xn — начало отрезка по х;
* xk — конец отрезка по х;
* xi — середина отрезка по х;
* epsy — требуемая точность вычислений по y (заданное приближение интервала [xn; xk] : xk — xn к нулю).

Тогда алгоритм метода бисекции можно записать в [псевдокоде](https://ru.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D0%BA%D0%BE%D0%B4_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2)) следующим образом:

1. Начало.
2. Ввод xn, xk, epsy.
3. Если F(xn) = 0, то Вывод (корень уравнения — xn).
4. Если F(xk) = 0, то Вывод (корень уравнения — xk).
5. Пока xk — xn > epsy повторять:
6. dx := (xk — xn)/2
7. xi := xn + dx;
8. если sign(F(xn)) ≠ sign(F(xi)), то xk := xi;
9. иначе xn := xi.
10. конец повторять
11. Вывод (Найден корень уравнения — xi с точностью по y — epsy).
12. Конец.

***Нахождение методом половинного деления всех корней уравнения f(x)=0 в заданном интервале изменения x (интервал принадлежит области определения функции и может содержать несколько корней).***

Для нахождения всех корней уравнения методом половинного деления необходимо в цикле   
по области определения функции выделять интервалы, в которых имеется ровно один корень, и вызывать для каждого интервала функцию нахождения корней методом половинного деления.

## Содержание отчета

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

- Постановка задачи;

* Разработка алгоритма;

- Кодирование (соответствующий раздел отчета называется «Текст программы»);

- Тестирование (соответствующий раздел отчета называется «Анализ результатов»).

В разделе «Постановка задачи» должен быть приведен текст задания.

Раздел «Разработка алгоритма» должен содержать следующую информацию:

* краткое словесное описание алгоритма (для 3-х методов);
* описание входных, выходных и вспомогательных данных с указанием их идентификаторов и типов;
* укрупненную схему алгоритма для пункта 3 задания;
* уточненной схемы ***одного из блоков***:
* Простая итерация (блок 0);
* Метод Ньютона (блок 1);
* Половинное деление(блок 2).

Номер блока равен остатку от деления порядкового номера студента в списке группы на 3 (блоки нумеруются, начиная с 0).

В разделе «Текст программы» должен быть приведен листинг программы, включающий ***необходимые комментарии***.

В разделе «Анализ результатов» должны быть приведены распечатки экранов и объяснение результатов для п.3 задания. Для экономии краски изображения экранов должны иметь белый фон.

Отчет должен быть распечатан на принтере на листах бумаги формата А4, скрепленных в левом верхнем углу с помощью степлера, и подписан исполнителем с указанием даты сдачи отчета преподавателю.

## Контрольные вопросы

1.Нарисуйте схему алгоритма нахождения корней уравнения f(x) = 0 методом простой итерации***.***

2. Нарисуйте схему алгоритма нахождения корней уравнения f(x) = 0 методом Ньютона.

3. Нарисуйте схему алгоритма нахождения корней уравнения f(x) = 0 методом половинного деления.

4. Приведите примеры объявления прототипа функции, определения и вызова функции.

5. Приведите примеры объявления указателя на функцию без использования оператора *typedef*.

6. Приведите примеры объявления указателя на функцию с использованием оператора *typedef*.

7. Как вызвать функцию через указатель?

8. Как можно передать функцию в качестве параметра в другую функцию и для чего это делается?

9. Напишите фрагменты программы нахождения всех корней уравнения *f(x) = 0* методом половинного деления, относящиеся к вызову функции:

- объявление прототипа функции;

- вычисление параметров функции;

- вызов функции с указанием её параметров.

## Рекомендуемые источники информации

# Лабораторная работа 5 Сортировка одномерного числового массива

## Цель работы

- разработка и использование функций, реализующих алгоритмы сортировки массивов методом выбора максимального (минимального) элемента и методом пузырькового всплытия;

- оценка быстродействия алгоритмов сортировки;

- создание шаблонов для разработанных функций сортировки;

- использование динамических массивов;

- использование шаблонов для сортировки по различным признакам массивов из структур, содержащих поля с данными различных типов.

- сравнение быстродействия алгоритмов в зависимости от размера и упорядоченности элементов массива.

Планируемое время выполнения работы - 6 часов.

## Задание

ЛР состоит из двух частей.  
 В первой части выполняется разработка и тестирование функций, реализующих алгоритмы сортировки массивов методом выбора максимального (минимального) элемента и методом пузырькового всплытия.  
 Во второй части создаются шаблоны разработанных в п.1 функций сортировки и с использованием шаблонов выполняется сортировка массива структур и сравнение быстродействия алгоритмов в зависимости от размера и упорядоченности элементов массива.

В Visual Studio решение, реализующее ЛР, должно состоять из двух проектов:  
 первый проект реализует часть1 задания, а второй – часть 2.

Часть 1.

1.1 Разработать функции для сортировки целочисленного числового массива методом выбора максимального (минимального) элемента и методом пузырькового всплытия.

В методе пузырькового всплытия цикл сравнений начинать с конца сортируемого массива (см. п.5 раздела «Указания по выполнению задания»).

Для сравнения быстродействия алгоритмов дополнительно включить в функции операторы для подсчета количества выполненных при сортировке сравнений и перестановок элементов массива.

1.2 Выполнить сортировку тестового массива по возрастанию и по убыванию значений элементов массива каждой из двух функций и распечатать отсортированный массив и количество сделанных при сортировке сравнений и перестановок элементов массива. *Распечатку результатов сортировки выполнять в функции main(), а данные для печати должны возвращаться из функций, выполняющих сортировку* (функция сортировки не должна печатать данные).

Часть 2.

2.1 Создать шаблоны для функций сортировки, разработанных в части 1 ЛР:

- один шаблон для сортировки методом пузырькового всплытия, начиная с конца массива;

- один шаблон для сортировки методом выбора максимального (минимального) элемента.

Шаблоны поместить в заголовочный файл и подключить его к проекту.

2.2 Создать структуры Date и Student и массив group из 10 элементов типа Student.

struct Date{

int day;

int month;

int year;

};

struct Student{

string name;

Date birthDay;

char id[6];//номер зачётной книжки  
};

2.3 Используя шаблоны, выполнить сортировку массива group по трем признакам:

- по возрастанию значения поля name,

- по убыванию значения поля birthDay,

- по возрастанию номера зачетной книжки.

2.4 Сравнить быстродействия алгоритмов сортировки в зависимости от размера и упорядоченности элементов массива. Для сравнения быстродействия алгоритмов используйте целочисленные массивы.

Результаты должны содержать числа сравнений и перестановок, выполненных каждой из функций в процессе сортировки массивов размером 100 и10000 чисел для различных состояний упорядоченности элементов массивов (см. п.7 раздела «Указания по выполнению задания»).

Возможность изменения длины массива реализуйте с помощью динамического массива, а для его инициализации используйте датчик случайных чисел (см. п.3.3 раздела “Примеры работы с массивами”).

Образец таблицы результатов сравнения алгоритмов сортировки приведен в п.8 раздела «Указания по выполнению задания». Элементы массивов не распечатывать.

Объясните результаты сравнения.

### Указания по выполнению задания.

1. Разработку программы начинайте с создания функции *int* *Menu().* Использование менюявляется удобным способом разделения программы на части. Это ускорит разработку и отладку программы и проверку преподавателем правильности выполнения пунктов задания ЛР.

2. Пример использования меню в программе.

int Menu(){

cout<<"================СОРТИРОВКА МАССИВОВ===================\n";

cout<<"\t1 - Исходный список группы студентов\n";

cout<<"\t2 – miniMax: Сортировка по убыванию даты рождения студента\n";

…

cout<<"\t5 - bubbleEnd: Сортировка по возрастанию id студента\n";

…

cout<<"\t8 - Выход\n";

int choice;

cout<<"Выберите действие\n"

cin>>choice;

while(cin.fail()){//исключение зацикливания в случае ввода недопустимых символов

cout<<"Ошибка ввода. Повторите ввод\n";

cin.clear() ;//сброс в потоке флага ошибки

cin.ignore(10,'\n');//извлечение из буфера потока введенных символов

cin>>choice;//повторный ввод

}

return choice;

}

Функция *main( )* с меню будет иметь следующий вид:

int main(){

…  
while(true){

switch(Menu()){

case 1:… break;//выполнение действия задания

case 2:… break;

…

case n: return 0;//завершение работы

default: cout<<”Недопустимое действие. Повторите выбор.\n”;   
 }

}

…

return 0;

}

3. Для отладки программы ***обязательно*** разработайте контрольный пример - ***статический*** массив из 10 элементов (см. п. 7.3.1 раздела “Примеры работы с массивами”). Значения элементов тестового массива и их количество должны быть минимально-достаточными, чтобы обеспечивать удобство контроля и полноту тестирования. Нужно предусмотреть возможность простого изменения размера и значений тестового массива в программе.

4. В процессе отладки используйте пошаговый режим выполнения программы и пошаговое сравнение значений в окне отладчика с результатами контрольного примера. Для ускорения отладки используйте «останов с условием».

5. В методе пузырькового всплытия цикл сравнений (внутренний цикл) можно начинать как с конца, так и с начала массива. Если массив не отсортирован, то эти варианты реализации метода равноценны. При работе с данными обычно приходится добавлять или удалять данные в уже отсортированный массив. Удаление данных выполняется путем сдвига «хвоста» массива влево, а новые элементы добавляются в ***конец*** массива и затем выполняется его сортировка. В этом случае первый вариант оказывается более эффективным.

6. Для того, чтобы один и тот же шаблон можно было бы использовать для сортировки массивов из различных элементов по различным условиям, для сравнения элементов сортируемых массивов в шаблоне нужно разработать разные функции сравнения. Эти функции должны иметь одинаковую сигнатуру(перегрузка), чтобы в шаблоне их можно было бы вызывать через указатель на эти функции

bool (\*cmp)(T obj1,T obj2), где

T – тип элементов в сортируемом массиве, который является параметром шаблона.

Этот указатель нужно добавить в список параметров в шаблон функции сортировки.

Примеры функций сравнения:

bool cmp1(int a, int b){ return a > b;} //для сортировки целочисленного массива по возрастанию;

bool cmp2(int a, int b){ return a < b;} ;} //для сортировки целочисленного массива по убыванию;

bool cmp3(char\* a, char\* b){ return strcmp(a , b)>0;} //для сортировки массива из символьных строк по возрастанию;

7. Состояния массива:

I – исходный со случайно расположенными числами,   
II – исходный массив, предварительно отсортированный по возрастанию,

III - исходный массив, предварительно отсортированный по убыванию.   
 Предварительно отсортированный массив представляет собой отсортированный случайный массив, в котором последний элемент заменен вручную на элемент, меньший его нулевого элемента (для массива, отсортированного по возрастанию) или на элемент, больший его нулевого элемента (для массива, отсортированного по убыванию). Таким образом имитируется добавление новых элементов в конец существующего отсортированного массива.

7. Образец таблицы сравнения результатов быстродействия алгоритмов сортировки.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Алгоритм | Размер массива | | | | | |
| 100 | | | | | |
| Состояние массива | | | | | |
| I | | II | | III | |
| compare | swap | compare | swap | compare | swap |
| bubbleEnd |  |  |  |  |  |  |
| minMax |  |  |  |  |  |  |

## 7.3 Примеры работы с массивами

Примеры этого раздела необходимо выполнить в случае, если к моменту выполнения ЛР вы ещё не освоили работу с массивами (они не являются частью ЛР).

При выполнении работы обратите внимание на следующие требования и рекомендации:

1. Размерность нединамического массива может быть только константой или константным выражением. Рекомендуется задавать размерность с помощью именованной константы.
2. Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности.
3. Автоматический контроль выхода индекса за границы массива не производится, поэтому программист должен следить за этим самостоятельно.
4. Указатель — это переменная, в которой хранится адрес области памяти.
5. Имя массива является указателем на его нулевой элемент.
6. Обнуления динамической памяти при ее выделении не происходит.
7. Память для динамического массива выделяется оператором new[]. Освобождение памяти, выделенной посредством оператора new[], выполняется с помощью оператора delete[].
8. *Перед выходом локального указателя из области его действия* следует освобождать связанную с ним динамическую память, так как при уничтожении указателя будет потеряна связь памяти с программой и она станет «мусором» (память останется занятой программой, но так как ее адрес потерян в связи с удалением указателя, то программа не сможет ее использовать).
9. Если количество элементов, которые должны быть введены в программу, известно до ее выполнения, определяйте массив в операторе описания переменных, причем лучше как локальную переменную, чем как глобальную.

Если количество элементов можно определить только во время выполнения программы, но до ввода элементов, создавайте динамический массив.

1. Алгоритмы сортировки массивов различаются по быстродействию и занимаемой памяти, причем эти характеристики зависят от упорядоченности сортируемого массива.

В случае простых переменных каждой области памяти для хранения одной вели­чины соответствует свое имя. Если требуется работать с группой величин одного типа, их располагают в памяти последовательно и дают им общее имя, а различают по порядковому номеру. Такая последовательность однотипных величин называ­ется массивом. Массивы, как и любые другие объекты, можно размещать либо с помощью опера­торов описания в сегментах данных или стека, либо в динамической области памя­ти с помощью операций выделения памяти.

При описании массива после имени в квадратных скобках задается количество его элементов (размерность), например int a[10]. Массив располагается в зависимо­сти от места его описания либо в сегменте данных (глобальный массив), либо в сегменте стека (локальный массив), и все инструкции по выделению памяти формирует компилятор до выполнения про­граммы. Вследствие этого размерность массива может быть задана только констан­той или константным выражением.

При описании массив можно инициализировать, то есть присвоить его элементам начальные значения, например:

int а[10] = {1, 1, 2, 2, 5, 100};

Если инициализирующих значений меньше, чем элементов в массиве, остаток мас­сива обнуляется, если больше — выдается сообщение об ошибке.

Элементы массивов *нумеруются с нуля*, поэтому максимальный номер элемента всегда на единицу меньше размерности массива. Автоматический *контроль выхода индекса за границы мас­сива не производится*, поэтому программист должен следить за этим самостоятельно. Для данного массива элементы имеют номера от 0 до 9. Номер элемента указыва­ется после его имени в квадратных скобках, например, *а[0], а[3]*.

### 7.3.1 Количество элементов между минимальным и максимальным

Написать программу, которая для целочисленного массива из ***N*** элементов опре­деляет, сколько положительных элементов располагается между его максималь­ным и минимальным элементами.

Запишем алгоритм словами в самом общем виде.

Определить, где в массиве расположены первые по порядку его максимальные и минимальные эле­менты, то есть найти их индексы.

Просмотреть все элементы, расположенные между ними. Если элемент массива больше нуля, увеличить счетчик элементов на единицу.

Перед написанием программы полезно составить тестовый пример, чтобы более наглядно представить себе алгоритм решения задачи. Ниже представлен пример массива из 10 чисел, в котором между максимальным и минимальным элементами расположены 3 положительных числа, и знаком ‘+’ обозначены искомые величины:

***6 -8 15 9 -1 3 5 -10 12 2***

***макс + + + мин***

Ясно, что порядок расположения элементов в массиве заранее не известен, и сна­чала может следовать как максимальный, так и минимальный элемент, кроме того, они могут и совпадать. Поэтому прежде чем просматривать массив в поисках ко­личества положительных элементов, требуется определить, какой из этих индек­сов больше. Запишем уточненный алгоритм:

1. Определить, где в массиве расположены его максимальный и минимальный  
элементы:

* Задать начальные значения для индексов максимального и минимального элементов (например, равные нулю, но можно использовать любые другие значения индекса, не выходящие за границу массива).
* Просмотреть массив, поочередно сравнивая каждый его элемент с ранее найденными максимумом и минимумом. Если очередной элемент больше ранее  
  найденного максимума, принять этот элемент за новый максимум (т. е. за­помнить его индекс). Если очередной элемент меньше ранее найденного минимума, принять этот элемент за новый минимум.

2. Определить границы просмотра массива для поиска положительных элемен­тов, находящихся между его максимальным и минимальным элементами:

* Если максимум расположен в массиве раньше, чем минимум, принять левую  
  границу просмотра равной индексу максимума, иначе — индексу минимума.
* Если максимум расположен в массиве раньше, чем минимум, принять правую  
  границу просмотра равной индексу минимума, иначе — индексу максимума.

3. Обнулить счетчик положительных элементов. Просмотреть массив в указанном диапазоне. Если очередной элемент больше нуля, увеличить счетчик на единицу.

*Для экономии времени при отладке рекомендуется значения элементов массива задавать путем инициализации, а размер массива вычислять с помощью оператора sizeof* *.*

#include <iostream >

using namespace std;

int main(){

system(”chcp 1251 > nul”);

int a[ ] = {6, 15, 9,-1, 3, 5,-10, 12, 2}; // данные контрольного примера

int n = sizeof(a)/sizeof(int);  
// нахождение максимального и минимального элементов массива.

int i, imax = 0, imin = 0;

for (i = 1; i < n; i++)

{

if (a[i] > a [imax]) imax = i;

if (a[i] < a[imin]) imin = i;

}  
// определение левой и правой границы интервала между максимальным и минимальным //элементами

cout <<"\n\t max= "<< a[imax]<< " min= "<< a[imin]; // отладка

int ibeg = imax < imin ? imax : imin;

int iend = imax < imin ? imin : imax;

cout <<"\n\t ibeg= "<< ibeg <<" iend= " << iend; // отладка  
//подсчет числа положительных элементов

int count;

for (count = 0, i = ibeg + 1; i < iend; i++)

if (a[i] > 0) count++;

cout << " Количество положительных " << count << endl;

system(“pause”);

return 0;

}

Программа содержит 3 структурных блока:

- нахождение максимального и минимального элементов массива;

- определение левой и правой границы интервала между максимальным и минимальным элементами;

- подсчет числа положительных элементов.

Для удобства отладки и тестирования программы после каждого блока выводятся результаты его работы.

Не пренебрегайте этим способом отладки и не жалейте времени, стараясь сделать эту печать хорошо читаемой, то есть содержащей необходимые пояснения и хорошо отформатированной. Это может сэкономить много времени при отладке программы. В программе использована управляющая последовательность \*t*, которая задает отступ при выводе на следующую позицию табуляции.

Массив просматривается, начиная с элемента, следующего за максимальным (мини­мальным), до элемента, предшествующего минимальному (максимальному). Ин­дексы границ просмотра хранятся в переменных *ibeg* и *iend*. В приведенной выше программе для определения их значений используется тернарная условная опера­ция. Можно поступить и по-другому; просматривать массив всегда от максимума к минимуму, а индекс при просмотре увеличивать или уменьшать в зависимости от их взаимного расположения.

Тестовых примеров для этой задачи должно быть по крайней мере три для случа­ев, когда:

*а [ imin ]* расположен левее *а [ imax];*

*a[imin]* расположен правее *a[imax];*

*a[imin]* и *a[imax]* совпадают.

Последняя ситуация имеет место, когда в массиве все элементы имеют одно и то же значение. Желательно также проверить, как работает программа, если *a[imin]* и *a[imax]* расположены рядом, а также в начале и в конце массива (граничные случаи). Элементы массива нужно задавать как положительные, так и отрицательные.

Часто бывает, что точное количество элементов в исходном массиве не задано, но известно, что оно не может превышать некое конкретное значение. В этом случае память под массив выделяется «по максимуму», а затем заполняется только часть этой памяти.

Память можно выделить либо с помощью оператора описания (в сте­ке или сегменте данных), либо с помощью операции new (в динамической области). Фактическое количество введенных элементов запоминается в переменной, которая затем участвует в орга­низации циклов по массиву, задавая его верхнюю границу. Этот подход является весьма распространенным. Рассмотрим небольшую, но полезную программу, в которой выполняется только считывание элементов массива с кла­виатуры и их вывод на экран:

#include <iostream>

using namespace std;

int main(){

system(chcp 1251>nul);

const int N = 100;

int a[N];

int kol\_a;

cout<< "Введите количество элементов\n";

cin >> kol\_a;

if (kol\_a > N ) {

cout <<" Превышение размеров массива\n"<< endl;

system(“pause”);

return 1;

}

for (int i = 0; i < kol\_a; i++)  
 cin >> a[i];

for (int i = 0; i < kol\_a; i++)   
 cout << a[i] << " ";

cout << endl;

system(“pause”);

return 0;

}

Несмотря на то, что значение константы *N* определяется «с запасом», надо обяза­тельно проверять, не запрашивается ли большее количество элементов, чем воз­можно. Привычка к проверке подобных, казалось бы, маловероятных случаев по­зволит вам создавать более надежные программы, а нет ничего более важного для программы, чем надежность.

### 7.3.2 Динамические массивы

Если до начала работы программы неизвестно, сколько в массиве элементов, в про­грамме следует использовать динамические массивы. Память под них выделяется с помощью операции ***new*** в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, на­зываемой указателем. Например:

int n = 10;

int \*a = new int[n];

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции new. Выделяется столько памяти, сколько необходимо для хранения ***n*** величин типа ***int***. *Величина* ***n*** *может быть переменной*.

*Обнуления памяти при ее выделении не происходит. Инициализировать динамический массив при объявлении нельзя.*

Обращение к элементу динамического массива осуществляется так же, как и к эле­менту обычного — например ***а[3].*** Можно обратиться к элементу массива и другим способом— **\**(а + 3)***. В этом случае мы явно задаем те же действия, что выполня­ются при обращении к элементу массива обычным образом. Рассмотрим их подробнее. В переменной-указателе ***а*** хранится адрес начала массива. ( Имя статического массива также является указателем на его первый элемент, только константным, то есть ему нельзя присвоить новое значение).

Для получения адреса третьего элемента к этому адресу прибавляется смещение 3. Операция сло­жения с константой для указателей учитывает размер адресуемых элементов, то есть на самом деле индекс умножается на длину элемента массива:

***а + 3* \* *sizeof(int).*** Затем с помощью операции **\*** (разадресация) выполняется выборка значения из указанной области памяти.

Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, необ­ходимо освободить ее с помощью операции *delete[ ]*, например:

delete [] a;

Размерность массива при этом не указывается.

*Квадратные скобки в операции delete [] при освобождении памяти из-под массива обяза­тельны. Их отсутствие может привести к неопределенному поведению программы (зависит от компилятора).*

Таким образом, время жизни динамического массива, как и любой динамической переменной, — с момента выделения памяти до момента ее освобождения. Область действия зависит от места описания указателя, через который производится рабо­та с массивом. Область действия и время жизни указателей подчиняются общим правилам. Как вы помните, локальная переменная при выходе из блока, в котором она описана, «теряется». Если эта перемен­ная является указателем и в ней хранится адрес выделенной динамической памя­ти, при выходе из блока эта память перестает быть доступной, однако не помечается как свободная и поэтому не может быть использована в дальнейшем. Это называет­ся утечкой памяти. Утечка памяти является распространенной и трудно обнаруживаемой ошибкой.

Пример утечки памяти:

…

{

int n;

cin >> n;

int \*pmas = new int[n];

}

…

После выхода из блока указатель *pmas* недоступен и память, на которую он указывал, становится «мусором».

### 7.3.3 Использование датчика случайных чисел.

#include <iostream>

using namespace std;

int iRandom( int a, int b ) // Генерирует случайные целые числа, распределенные в //интервале [a,b] по равномерному закону.

// rand() – программный датчик случайных целых чисел, равномерно распределенных в //интервале [0, RAND\_MAX]. RAND\_MAX = 215-1=32767

{

return a + (( b - a +1) \* rand()) / RAND\_MAX ; //целочисленное деление

}

double dRandom( int a, int b ) // Генерирует случайные вещественные числа, //распределенные в интервале [a,b] по равномерному закону.

{

return a + ( b - a ) \* rand() / (double)RAND\_MAX; // деление с плавающей точкой

}

void main()

{//При повторном запуске программы будет выдаваться одна и та же последовательность //чисел.

/\*Чтобы последовательность не повторялась, при запуске программы перед вызовом функции iRandom( a,b ) один раз нужно выполнить операторы:

srand( (unsigned int) time( NULL ) ); // установить начальное значение – текущее время

rand( ); //сброс первого числа, чтобы не повторялось, пока srand не изменится

\*/

const int n = 13; // количество элементов в массиве

int a[n], i; // массив целых чисел

for( i = 0; i < n; i++ ) // цикл инициализации массива

{ a[i] = iRandom( 1, 9 ); // целая случайная величина

cout<<”a[“<<i<<”]=”<<a[i]<<endl;

}

}

## 7.4 Содержание отчета

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

- Постановка задачи;

* Разработка алгоритма;

- Кодирование (соответствующий раздел отчета называется «Текст программы»);

- Тестирование (соответствующий раздел отчета называется «Анализ результатов»).

В разделе «Постановка задачи» должен быть приведен текст задания.

Раздел «Разработка алгоритма» должен содержать следующую информацию:

* краткое описание алгоритма (для 2-х методов сортировки);
* описание входных, выходных и вспомогательных данных с указанием их идентификаторов и типов;
* схему алгоритмов, состоящую из двух частей: общей (укрупненной) схемы и уточненной схемы ***одного из блоков***:
* Метод пузырькового всплытия (блок 0);
* Метод минимакса (блок 1);

Номер блока равен остатку от деления порядкового номера студента в списке группы на 2.

В разделе «Текст программы» должен быть приведен листинг программы, включающий ***необходимые комментарии***.

В разделе «Анализ результатов» должны быть приведены распечатки экранов и таблица результатов для п.2 задания.   
Для экономии краски изображения экранов должны иметь белый фон.

Отчет должен быть распечатан на принтере на листах бумаги формата А4, скрепленных в левом верхнем углу с помощью степлера, и подписан исполнителем с указанием даты сдачи отчета преподавателю.

## 7.5 Контрольные вопросы

1. В чем отличие алгоритмов сортировки методом выбора и методом пузырькового всплытия?
2. Объясните результаты сравнения быстродействия алгоритмов, полученные в ЛР.
3. Для чего предназначены операторы *return, break* и *continue*?
4. Почему массив должен состоять из элементов только одного типа?
5. Почему массив должен располагаться в непрерывной памяти?
6. Напишите фрагмент программы с динамическим массивом, приводящий к утечке памяти.
7. Что понимается под понятием «надежность программы»? Как можно повысить надежность программы?
8. Назовите 4 способа передачи результатов работы функции в вызывающую программу.
9. Для чего и как используются шаблоны функций?
10. Какие ограничения накладывает использование шаблона на генерируемые функции?

## 7.6 Рекомендуемые источники информации

1. Р. Лафоре. Объектно-ориентированное программирование в С++. Издательство ПИТЕР, 2004 г.

1. Н. Вирт. Алгоритмы+Структуры данных=Программы, «Мир», 1985 г.
2. <http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.
3. Г. Шилдт. Полный справочник по С++, 4-е издание, Издательский дом «Вильямс», 2006 г

# Лабораторная работа 6. Численное интегрирование функций

## Цель работы.

На примере разработки программы для численного интегрирования функции с заданной точностью методом прямоугольников и методом трапеций освоить следующие приемы программирования:

- передача в функцию параметров «по значению» и «по адресу»;

- передача в функцию другой функции в виде параметра;

- передача одномерных массивов в функцию;

- объединение разнородных данных в структуру;

- использование массивов из элементов типа структура;

- повышение эффективности использования вложенных циклов.

Планируемое время выполнения работы - 8 часов.

## Задание.

1. Численное интегрирование функции с заданной точностью методом прямоугольников.

Вычислить определённый интеграл в пределах от ***a*** до ***b*** для четырех функций f1 = x, f2 = sin( 22 \* x ), f3 = x4 и f4 = arctg(x).

Вычисление интеграла оформить в виде функции IntRect.

Вычисления выполнить для пяти значений точности: 0.01, 0.001, 0.0001, 0.00001 и 0.000001.

Исследовать быстродействие алгоритма в зависимости от подынтегральной функции и требуемой точности (быстродействие алгоритма можно оценить числом элементарных прямоугольников ***n,*** при котором достигнута требуемая точность интегрирования).

Результаты представить в виде 5 таблиц, по одной таблице для каждого значения точности. В каждой таблице выводить данные для всех четырех функций.

Для печати таблицы результатов использовать приведенную в приложении 2 функцию

void PrintTabl(I\_print i\_prn[],int k)

Здесь i\_prn[] – массив структур типа I\_print размерностью k.

Для ее использования нужно создать и заполнить массив из структур типа I\_print и передать его в функцию в качестве параметра.

Вид таблицы приведен в Приложении 1.

Для использования в функции PrintTabl( ) символов псевдографики заменитете в ней кодировочную таблицу Windous 1251 на кодировочную таблицу MS Dos 866 (так как это глобальная переменная, то перед выходом из функции нужно восстановить кодировочную таблицу Windous 1251).

2. Выполнить п.1, используя для интегрирования метод трапеций. Вычисление интеграла оформить в виде функции IntTrap.

Для печати таблиц результатов использовать ту же функцию, что и в методе прямоугольников.

## Рекомендации по выполнению работы.

Задача вычисления определенного интеграла формулируется следующим образом: вычислить для подынтегральной функции *f(x)* при заданных значениях пределов интегрирования ***a, b*** и требуемой точности ***eps***.

Так как в лабораторной работе требуется выполнять вычисление интеграла для четырех функций, для пяти значений точности для каждой функции и двумя методами, то для сокращения объема программы следует использовать циклы, а для обеспечения возможности реализации циклов обрабатываемые данные нужно хранить в массивах (массив указателей на функции, массив значений точности, массив структур для хранения и печати результатов вычислений).

Разработка алгоритма и отладка такой программы существенно упрощается, если ее выполнять по шагам, начиная с укрупненной схемы, которая отражает общую структуру программы и включает определение типов переменных, функций, массивов и циклы с их использованием. Эта программа будет уточняться в процессе разработки программы для выполнения лабораторной работы. Вместо функций интегрирования на первом шаге следует использовать “заглушки”, то есть функции, имеющие такой же прототип, что и функции интегрирования, но которые при вызове вместо реализации метода возвращают постоянные тестовые значения.

Кодирование и отладку функций численного интегрирования следует выполнять после того, как будет разработана и отлажена “тестовая” программа, которая для контроля тестовых данных использует функцию печати PrintTabl(I\_print i\_prn[],int k).

### Метод прямоугольников*.*

При численном интегрировании площадь под кривой заменяется суммой площадей «элементарных» прямоугольников с основанием ∆x и высотой, проведенной из середины основания.

Формула приближенного значения определенного интеграла представляется в виде

,

где: *xi = a + Δx/2 + (i-1)\*Δx; N -* число элементарных прямоугольников*.*

Для уменьшения объема вычислений множитель Δx следует вынести за знак суммы. Тогда в цикле нужно выполнять только суммирование, а затем полученную сумму один раз умножить на *Δx*.

Для оценки погрешности вычисления интеграла на практике используют правило Рунге. Суть правила состоит в том, что выполняют вычисление интеграла с двумя разными шагами изменения переменной *х*, а затем сравнивают результаты и получают оценку точности. Наиболее часто используемое правило связано с уменьшением *Δx* в два раза на каждом шаге вычисления интегральной суммы.

Для методов прямоугольников и трапеций погрешность R*Δx/2* вычисления интеграла с шагом *Δx/2* оценивается следующей формулой:

|R*Δx/2*| =, (1)

где I*Δx/2* – значение интеграла, вычисленное с шагом *Δx/2*; I*Δx* – значение интеграла, вычисленное с шагом *Δx.*Деление *Δx/2* продолжается до тех пор, пока погрешность |R*Δx/2*| не станет меньше ***eps.***

Для вычисления интеграла с точностью ***eps***используется метод итераций с двумя вложенными циклами. Во внутреннем цикле находят значение определенного интеграла с шагом *Δx/2*.

Во внешнем цикле производится сравнение значений интегральных сумм, вычисленных с шагами *Δx* (вычислена на предыдущем шаге) и *Δx/2* соответственно. Если требуемая точность не достигнута, то число разбиений удваивается, а в качестве предыдущего значения интеграла берут текущее и вычисление интеграла выполняется при новом числе разбиений.

Для организации итерационного цикла перед входом в цикл нужно определить начальные значения *Δx* и I*Δx*.

Вычисление интеграла оформить в виде функции IntRect, формальными параметрами которой являются:

*f* - имя интегрируемой функции,

*a, b* – границы интервала интегрирования,

*eps* – требуемая точность,

*n* – число прямоугольников, при котором достигнута требуемая точность (выходной).

Функция возвращает значение интеграла.

Прототип функции:

double IntRect(TPF f,double a,double b,double eps,int& n);

Здесь:

TPF – имя типа указателя на подынтегральную функцию, определяемый пользователем:

typedef double (\*TPF)(double);

После выполнения этого оператора в программе вместо вместо типа

double (\*)(double) - указатель на функцию, которая возвращает double и имеет один параметр типа double - можно писать просто TPF, то есть TPF становится синонимом double (\*)(double).

Для хранения и печати результатов вычислений используйте структуру, элементами которой являются наименование функции, значения интеграла (точное и вычисленное в виде суммы) и число «элементарных» прямоугольников ***n***, при котором достигнута требуемая точность. Точные значения, полученные аналитически, нужны для оценки правильности результатов численного интегрирования.

### Метод трапеций.

Алгоритм метода трапеций аналогичен алгоритму метода прямоугольников, только площадь элементарной трапеции вычисляется по формуле: Sт=dx\*(f(x)+f(x+dx))/2.

Для повышения эффективности алгоритма используют тот факт, что значения функции на границах внутренних отрезков при вычислении интегральной суммы используются дважды (как нижнее основание одной трапеции и как верхнее основание соседней трапеции). Значения функции на границах интервала [a,b] – ( f(a) и f(b) ) используются только один раз, но так как они не зависят от *x,* то их полусумму можно вычислить вне цикла суммирования.

Прототип функции:

double IntTrap(TPF f,double a,double b,double eps,int& n);

### Формулы для вычисления точных значений интеграла:

=(b\*b - a\*a)/2.0;

*sin(22x)dx*=(cos(a\*22.0) - cos(b\*22.0))/22.0;

*x4dx*=(b\*b\*b\*b\*b - a\*a\*a\*a\*a)/5.0;

=b\*atan(b) - a\*atan(a) - (log(b\*b+1) - log(a\*a+1))/2.0;

### Примеры передачи в функцию в качестве параметров одномерных массивов и имен функций.

Массивы и функции передаются в функцию через указатели.

Имя массива является указателем на его нулевой элемент. Указатель «ничего не знает» о длине массива и длина массива должна передаваться в функцию как параметр.

Имя функции указывает на первую команду кода функции, и в этом качестве является указателем на функцию.

*Передача одномерных массивов в функцию*

#include <iostream.h>

int sum(int \*a,int n);

int main() {

int n;

int a[]={1,2,3,4,5,6,7,8};

n=sizeof(a)/sizeof(int);// Определение размерности инициализированного

// массива

cout<<"n="<<n<<endl;

cout << sum(a,n) <<"\n";

return 0;

}

int sum(int\* a,int n) //В функцию передаются указатель на начало массива

//(имя массива a) и его размерность (n)

{

int i,s=0;

int k=sizeof(a); //k – размер указателя (4 байта), а не размер массива в байтах

cout<<"k="<<k<<endl;

for (i = 0; i < n; i++)

s += a[i];

return s;

}

*Передача имен функций в качестве параметров*

#include <iostream>

using namespace std;

/\*для удобочитаемости программы определяется новый тип (тип пользователя):

***PF*** - указатель на функцию, которая в этом примере имеет один параметр типа int и не возвращает никакого значения\*/

typedef void (\*PF)(int);

//Определение функции f1

void f1(PF pf) //функция получает в качестве параметра указатель типа PF

{

pf(5); //вызов функции через указатель

}

void f(int i)

{

cout << i<<endl;

}

int main()

{

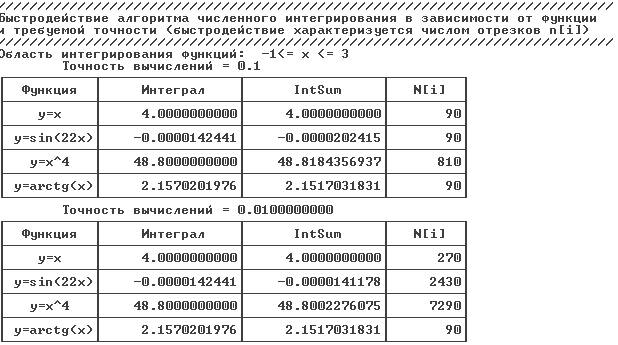
f1(f); //Функция выведет на экран число 5

return 0;

}

Приложение 1.

### Пример вывода таблицы результатов



Приложение 2.

### Функция для печати таблицы результатов

struct I\_print{ //данные для печати результатов интегрирования

char\* name;//название функции

double i\_sum; //значение интегральной суммы

double i\_toch; //точное значение интеграла

int n; //число разбиений области интегрирования

//при котором достигнута требуемая точность

};

void PrintTabl(I\_print i\_prn[],int k)

{

system("chcp 866>nul");

const int m=4;//число столбцов таблицы

int wn[m]={12,18,18,10};//ширина столбцов таблицы

const char \*title[m]={"Function","Integral","IntSum","N "};

int size[m];

for(int i=0;i<m;i++)

size[i]=strlen(title[i]);

//шапка таблицы

cout<<char(218)<<setfill(char(196));

for(int j=0;j<m-1;j++)

cout<<setw(wn[j])<<char(194);

cout<<setw(wn[m-1])<<char(191)<<endl;

cout<<char(179);

for(int j=0;j<m;j++)

cout<<setw((wn[j]-size[j])/2)<<setfill(' ')<<' '<<title[j]

<<setw((wn[j]-size[j])/2)<<char(179);

cout<<endl;

for(int i=0;i<k;i++)

{//заполнение таблицы

cout<<char(195)<<fixed;

for(int j=0;j<m-1;j++)

cout<<setfill(char(196))<<setw(wn[j])<<char(197);

cout<<setw(wn[m-1])<<char(180)<<setfill(' ')<<endl;

cout<<char(179)<<setw((wn[0]-strlen(i\_prn[i].name))/2)<<' ' <<i\_prn[i].name <<setw((wn[0]-strlen(i\_prn[i].name))/2)<<char(179);

cout<<setw(wn[1]-1)<<setprecision(10)<<i\_prn[i].i\_toch<<char(179)

<<setw(wn[2]-1)<<i\_prn[i].i\_sum<<char(179)

<<setw(wn[3]-1)<<i\_prn[i].n<<char(179)<<endl;

}

//низ таблицы

cout<<char(192)<<setfill(char(196));

for(int j=0;j<m-1;j++)

cout<<setw(wn[j])<<char(193);

cout<<setw(wn[m-1])<<char(217)<<endl;

//восстановление первоначальных значений

cout<<setprecision(6)<<setfill(' ');

system("chcp 1251>nul");

}

## Содержание отчета

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

- Постановка задачи;

- Разработка алгоритма;

- Кодирование (соответствующий раздел отчета называется «Текст программы»);

- Тестирование (соответствующий раздел отчета называется «Анализ результатов»).

В разделе «Постановка задачи» должен быть приведен текст задания и согласованные с преподавателем (заказчиком) уточнения, если они требуются.

Раздел «Разработка алгоритма» должен содержать следующую информацию:

* краткое описание (обоснование) алгоритма,
* описание входных, выходных и вспомогательных данных с указанием их идентификаторов и типов,
* схему алгоритма, состоящую из двух частей: общей (укрупненной) схемы и уточненной схемы одной из функций:
  + double IntRect(TPF f,double a,double b,double eps,int& n);
  + double IntTrap(TPF f,double a,double b,double eps,int& n).

(Номер блока равен остатку от деления порядкового номера студента в списке группы на 2 плюс 1.)

В разделе «Текст программы» должен быть приведен листинг программы, включающий необходимые комментарии.

В разделе «Анализ результатов» должны быть приведены распечатки экранов для обоих методов численного интегрирования при eps = 0.00001 и eps = 0.000001. Для экономии краски изображения экранов должны иметь белый фон.

Отчет должен быть распечатан на принтере на листах бумаги формата А4, скрепленных в левом верхнем углу с помощью степлера, и подписан исполнителем с указанием даты сдачи отчета преподавателю.

## Контрольные вопросы

* Определение и использование функций.
* Обмен данными с функциями через возвращаемые значения, параметры и глобальные переменные. Передача параметров по ссылке и по значению.
* Перегрузка функций.
* Указатели на функции и их использование (объявление, передача указателей на функцию в качестве параметров и вызов функций с помощью указателя в теле функции) для динамического выбора функций во время выполнения программы.
* Передача одномерных массивов в функцию;
* Объединение разнородных данных в структуру;
* Как реализуется сокращение времени вычисления интегралов в методе прямоугольников?
* Как реализуется сокращение времени вычисления интегралов в методе трапеций?
* Использование массивов из элементов типа структура.
* Нарисовать схему алгоритма численного интегрирования методом прямоугольников.
* Нарисовать схему алгоритма численного интегрирования методом трапеций.

## Рекомендуемые источники информации

1. Р. Лафоре. Объектно-ориентированное программирование в С++. Издательство ПИТЕР, 2004 г.

2. Н. Вирт. Алгоритмы+Структуры данных=Программы, «Мир», 1985 г.

3. <http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.

4. Г. Шилдт. Полный справочник по С++, 4-е издание, Издательский дом «Вильямс», 2006 г

# Лабораторная работа 7 Обработка и печать числовой матрицы

## Цель работы

- создание двухмерных динамических массивов;

-использование файлов для хранения матриц;

- форматированный вывод матриц на экран;

- обработка матриц;

- передача двухмерных массивов в функцию;

- разработка шаблона функции печати числовых матриц;

- доступ к элементам матрицы через указатели и с помощью индексов;

- освоение технологии структурного программирования.

Планируемое время выполнения работы - 6 часов.

## Задание

2.1 Разработать функцию для вывода на экран матрицы double matr[N][M] для распечатки двухмерных массивов. Размерность матрицы может быть произвольной.

Функция должна обеспечивать удобное для пользователя отображение матрицы на экране в (фиксированном (fixed) или )научном (scientific) формате с требуемой точностью ***n*** знаков после запятой (устанавливается setprecision(n)). Целые числа также выводятся в научном формате. Матрица,(тип формата,) число знаков после запятой должны передаваться в функцию в виде параметров (число знаков в целой части для научного формата всегда равно 1).

*Указание:* В зависимости от размера матрицы и требуемого числа печатаемых знаков после запятой, функция должна вычислить ширину поля вывода одного элемента матрицы ***w*** (при печати используется в операторе setw(w) ) и определить число столбцов матрицы (ncol), которое поместится в одной строке на экране.

Строка на экране вмещает 80 символов и все столбцы матрицы могут не поместиться в одной строке. В этом случае печать матрицы следует выполнять в несколько рядов по ncol столбцов в каждом (кроме последнего ряда). В каждом ряду печатайте все строки столбцов этого ряда. Над каждым столбцом напечатайте его номер.

2.2 Используйте разработанную в п. 2.1 функцию для печати квадратной матрицы А размером N\*N (где N вводится с клавиатуры).

- все элементы главной диагонали матрицы А равны 1;

- элементы, лежащие выше главной диагонали, вычисляются по формуле

**A i,j = xi / (j!)i ,**

а элементы, лежащие ниже главной диагонали, по формуле

**A i,j = (-x)i / (j!)i**, где **i,j =1,2,…,N**; **x=1**.

Для вычисления значений элементов матрицы используйте ***рекуррентные соотношения*** (без факториалов и степеней).

Создание и заполнение матрицы реализуйте в виде ***отдельных функций***.

(2.3 Не изменяя кода функции печати матрицы, распечатать матрицу в «научном» формате и в формате с фиксированной точкой с требуемой точностью.)

2.3 Распечатать с помощью разработанной функции целочисленную статическую (заданную с помощью оператора описания) матрицу размером **B[10][10].** Для передачи матрицы в функцию печати создайте вспомогательный массив указателей на строки матрицы **B** . Значения элементов матрицы **В** определяются соотношением: **B[i][j]=i\*10+j**.

2.5 Создайте шаблон для функции печати числовых матриц и выполните п.2.2 и 2.3 с использованием шаблона;

2.5 Вставьте в программу и объясните результаты выполнения следующих операторов

для матрицы В[10][10]:

cout<<B<<" "<<B[0]<<" "<<B[2]<<endl;

cout<<B[0][0]<<" "<<\*\*B <<" "<<\*B[0]<<endl;

cout<<\*(\*(B+1))<<" "<<\*B[1]<<endl;

cout<<\*(B[0]+1)<<" " <<\*(\*B+1)<<endl;

cout<<B[0][20]<<" "<<\*(B[0]+20)<<" "<<\*B[2]<<endl;

## Рекомендации по выполнению работы

При работе с матрицами нужно всегда помнить, что в С++ нет объектов типа многомерный массив. Двухмерный массив - это одномерный массив, каждый элемент которого является одномерным массивом.

Прежде чем приступать к выполнению задания прочитайте рекомендации по выполнению работы и выполните по шагам программу сортировки строк матрицы, приведенную в п. 9.3.3 (в отчет программу не включать).

### Создание двухмерных динамических массивов.

В динамической области памяти можно создавать двухмерные массивы с помощью операции new.   
Обращение к элементам динамических массивов производится точно так же, как к элементам «обычных», с помощью конструкции вида a[i][j].

Универсальный способ выделения памяти под двухмерный массив, когда обе его размерности задаются на этапе выполнения программы, при­веден ниже:

int nrow, ncol;

cout << " Введите количество строк и столбцов :";

cin >> nrow >> ncol;

int \*\*a = new int \*[nrow]; // 1

for(int i = 0; i < nrow; i++) // 2

a[i] = new int [ncol]; // 3

Для того чтобы понять, отчего динамические массивы описываются именно так, нужно разобраться в механизме индексации элемента массива. Поскольку для доступа к элементу массива применяется две опе­рации разадресации, то переменная, в которой хранится адрес начала массива, долж­на быть указателем на указатель (указывать на начало одномерного массива).

В операторе 1 объявляется переменная типа «указатель на указатель на **int**» и вы­деляется память под массив указателей на строки массива (количество строк — **nrow**). В операторе 2 организуется цикл для выделения памяти под каждую строку массива. В операторе 3 каждому элементу массива указателей на строки присваи­вается адрес начала участка памяти, выделенного под строку двумерного массива. Каждая строка состоит из **ncol** элементов типа **int** (на рис. 1 nstb=ncol).

Освобождение памяти из-под массива с любым количеством измерений выполняется с по­мощью операции **delete [],** например: delete [] a;

5

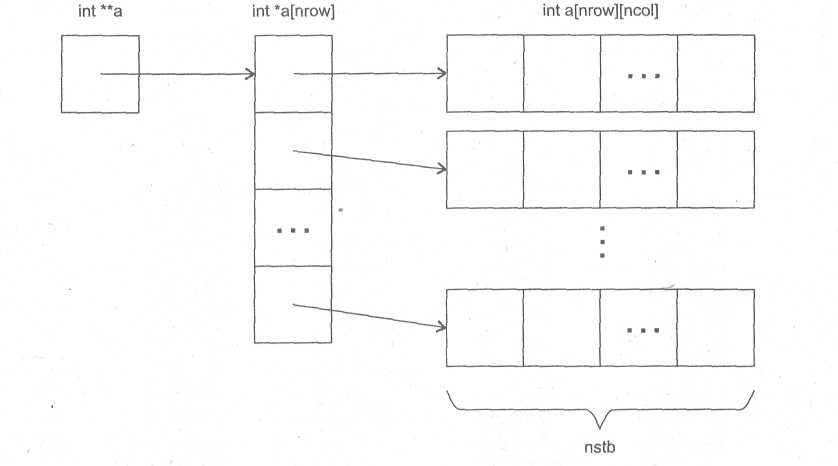


Рис. 1. Схема динамической области памяти, выделяемой под массивы

### Передача многомерного массива в функцию с помощью параметров.

При необходимости передать в функцию многомерный массив с помощью параметра возникают неудобства, связанные с отсутствием в Си++ объектов типа многомерный массив. Если мы описываем массив с несколькими индексами, например,

**double arr[6][4][2];**

то это не трехмерный массив, а одномерный массив с именем arr, состоящий из **6** элементов, каждый из которых имеет тип **double [4][2].** В свою очередь, каждый из этих элементов есть одномерный массив из четырех элементов типа **double [2].** И, наконец, каждый из этих элементов является массивом из двух элементов типа **double**.

Очевидное и неверное решение при попытке передать в функцию матрицу – определить её заголовок следующим образом:

**void func(double x[][], int n)**

Здесь **n** – предполагаемый порядок квадратной матрицы; **double x[][]** – попытка определить двухмерный массив с заранее неизвестными параметрами. На такую попытку транслятор ответит сообщением об ошибке:

Error…: Size of type is unknown or zero.

Вспомним – массив всегда одномерный, а его элементы должны иметь известную и фиксированную длину. В массиве **double x[][]** не только не известно количество элементов одномерного массива (это допустимо и их можно передать параметром **int n),**

но ничего не известно о размерах этих элементов. Допустимое с точки зрения синтаксиса языка Си++ решение - **void func(double x[][4], int n).**

Нежизненность такого решения – необходимость фиксации второй размерности матрицы.

Указанные ограничения на возможность применения многомерных массивов в качестве параметров функции можно обойти двумя путями.

Первый путь – подмена многомерного массива, например, **double x[3][4]** одномерным **double x[12]** и имитация внутри функции доступа к нему как к многомерному массиву.

Второй путь – использование вспомогательных одномерных массивов указателей на массивы. Такой массив указателей на строки матрицы используется при создании динамических массивов.

Помните, что если размерность массива явно не указана, то в функцию с помощью параметров можно передавать только одномерные массивы.

### Пример разработки программы сортировки строк матрицы

*Написать программу, которая упорядочивает строки прямоугольной целочислен­ной матрицы по возрастанию сумм их элементов. Исходная матрица хранится в файле с именем "input.txt", расположенном в текущей папке программы.*

Давайте на этом примере формализуем общий порядок создания структурированной про­граммы. Этому порядку полезно следовать при решении даже простейших задач.

**I. *Исходные данные, результаты и промежуточные величины.*** Как уже не­однократно упоминалось, начинать решение задачи необходимо с четкого описа­ния того, что является ее исходными данными и результатом и каким образом они будут представлены в программе.

*Исходные данные.* Поскольку размерность матрицы неизвестна, придется исполь­зовать динамический массив для хранения элементов матрицы. Числа строк и столбцов матрицы находятся в начале файла с элементами матрицы. Ограничимся элементами типа int.

*Результат.* Результатом является та же матрица, но упорядоченная. Это значит, что нам не следует заводить для результата новую область памяти, а необходимо упорядочить матрицу на том же месте. В данной задаче такое требо­вание может показаться излишним, но в общем случае, когда программист работа­ет в команде и должен передавать результаты коллеге, это важно. Представьте себе ситуацию, когда коллега думает, что получил от вас упорядоченную матрицу, а на самом деле вы сформировали ее в совершенно другой области памяти.

*Промежуточные величины.* Кроме конечных результатов, в любой программе есть промежуточные, а также служебные переменные. Следует выбрать их тип и спо­соб хранения.

Очевидно, что, если требуется упорядочить матрицу по возрастанию сумм элемен­тов ее строк, эти суммы надо вычислить и где-то хранить. Поскольку все они по­требуются при упорядочивании, их надо записать в массив, количество элементов которого соответствует количеству строк матрицы, а i-й элемент содержит сумму элементов i-й строки. Количество строк заранее неизвестно, поэтому этот массив также должен быть динамическим.

После того как выбраны структуры для хранения данных, можно подумать и об алгоритме, так как алгоритм зависит от того, каким образом представлены данные. В свою очередь, выбор алгоритма может потребовать изменения структуры данных. Всегда нужно иметь ввиду, что «программа = алгоритм + структура данных».

***II.* *Алгоритм работы программы.*** Для сортировки строк воспользуемся одним из самых простых методов — методом выбора. Он состоит в том, что из массива выбирается наименьший элемент и меняется местами с первым элементом, затем рассматриваются элементы, начиная со второго, и наименьший из них меняется местами со вторым элементом и так далее n - 1 раз. Одновременно с обменом элементов массива выполняется и обмен значе­ний двух соответствующих строк матрицы (перестановка строк).

Алгоритм сначала записывается в виде текста (например, так, как это сде­лано выше). Пренебрегать словесным описанием не следует, потому что процесс формулирования на естественном языке полезен для более четкого понимания задачи. При этом надо стремиться разбить алгоритм на простую последователь­ность шагов. Например, любой алгоритм можно первоначально разбить на этапы ввода исходных данных, вычислений и вывода результата.

Вычисление в данном случае состоит из двух шагов: формирование сумм элемен­тов каждой строки и упорядочивание матрицы. Упорядочивание состоит в выборе наименьшего элемента и обмене с первым из рассматриваемых.

Сложные алгоритмы (разветвленные алгоритмы и алгоритмы с циклами) полезно представить в виде обобщенной блок-схемы и перед кодированием выполнить проверку правильности работы алгоритма на тестовых примерах.

***III. Кодирование***. Когда алгоритм полностью прояснился, можно переходить к написанию про­граммы.Одновременно с этим продумываются и подготавливаются тестовые примеры. Не ленитесь придумать переменным понятные имена и сразу же при написании аккуратно форматировать текст программы, чтобы по положению оператора было видно, на каком уровне вложенности он находится. Функционально завершенные части алгоритма отделяются пустой строкой, комментарием или хотя бы комментарием вида

//-------------------------------------------------------------

***IV. Отладка***. При написании программы рекомендуется всегда включать в нее промежуточную печать вычисляемых величин в удобном для восприятия формате. Это простой и надежный спо­соб контроля хода выполнения программы.

Не нужно стремиться написать сразу всю программу. Сначала пишется и отлажи­вается фрагмент, содержащий ввод исходных данных. Затем промежуточную пе­чать можно убрать и переходить к следующему функционально законченному фрагменту алгоритма. Для отладки полезно выполнять программу по шагам с на­блюдением значений изменяемых величин и сравнением их с контрольным примером.

Программа сортировки строк матрицы.

#include <fstream>

#include <iomanip>

using namespace std;

int main()

{

ifstream fin("input.txt", ios::in);

if (fin.fail())

{ cout <<" Файл input.txt не найден " << endl;

return 1;

}

int nrow,ncol;

fin>>nrow>>ncol; // ввод размерности массива

int i, j;

// выделение памяти под вспомогательный массив указателей на строки

int \*\*a = new int \*[nrow];

// выделение памяти под строки матрицы

//и инициализация вспомогательного массива указателей на строки

for(i = 0; i < nrow; i++)

a[i] = new int [ncol];

//ввод массива

for (i = 0; i < nrow; i++)

for (j = 0; j < ncol; j++)

fin >> a[i][j];

// формирование массива сумм элементов строк

int \*sum = new int [nrow];

for (i =0; i < nrow; i++)

{

sum[i] = 0;

for (j = 0; j < ncol; j++)

sum[i] += a[i][j];

}

// контрольный вывод

for (i = 0; i < nrow; i++)

{

for (j = 0; j < ncol; j++)

cout << setw(4) << a[i][j] << " ";

cout << "| " << sum[i] << endl;

}

cout << endl;

int buf\_sum;

int nmin, buf\_a;

for (i = 0; i < nrow - 1; i++) // упорядочивание

{

nmin = i;

for (j = i + 1; j < nrow; j++)

if (sum[j] < sum[nmin]) nmin = j;

int \* prow=a[i];//перестановка строк

a[i]=a[nmin];

a[nmin]=prow;

buf\_sum = sum[i];

sum[i] = sum[nmin];

sum[nmin] = buf\_sum;

}

for (i = 0; i < nrow; i++)

{ // вывод упорядоченной матрицы

for (j = 0; j < ncol; j++) cout <<setw(4) << a[i][j] << " ";

cout << endl;

}

return 0;

}

В программе используются две буферные переменные:

buf\_sum - для обмена значений сумм элементов двух строк массива,

buf\_a - для обмена значений двух элементов массива

Эти переменные того же типа, что и элементы массива.

Данные читаются из файла. Этот способ ввода является предпочтительнее стандартного ввода с клавиатуры по двум причинам:

1. Не нужно повторять ввод при каждом запуске программы.
2. При формиро­вании файла легче продумать, какие значения лучше взять для исчерпывающего тестирования программы.

В данном случае для первого теста следует подготовить массив не менее чем из четырех строк с небольшими значениями элементов для того, чтобы можно было в уме проверить, правильно ли вычисляются суммы.

Ввод размерности массива и его элементов выполняется из файла input.txt, рас­положенного в том же каталоге, что и программа, а результаты выводятся в файл output. txt.

В программе определены объект fin класса входных файловых потоков и объект fout класса выходных файловых потоков. Файловые потоки описаны в заголовочном файле <fstream>. Работа с этими объектами аналогична работе со стандартными объектами cin и cout, то есть можно пользоваться теми же операци­ями помещения в поток << и извлечения из потока >>.

Предполагается, что файл с именем input.txt находится в том же каталоге, что и текст программы, иначе следует указать полный путь, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение:

ifstream fin("input.txt", ios::in | ios::nocreate);

Обратите внимание, что для контроля вместе с исходным массивом рядом с каж­дой строкой выводится сумма ее элементов, отделенная вертикальной чертой.

Дополнительно следует прове­рить, правильно ли упорядочивается массив из одной и двух строк и столбцов, поскольку многие ошибки при написании циклов связаны с неверным указанием их граничных значений.

### Основные правила работы с двухмерными массивами

1. В массивах, определенных с помощью операторов описания, обе размерности должны быть константами или константными выражениями.
2. Массив хранится по строкам, каждая строка хранится в непрерывной области памяти.
3. Первый индекс всегда представляет собой номер строки, второй — номер столбца. Каждый индекс может изменяться от 0 до значения соответствующей раз­мерности, уменьшенной на единицу.
4. При описании массива можно в фигурных скобках задать начальные значения его элементов.
5. Для выделения динамической памяти под массив, в котором все размерности переменные, используются циклы.
6. Освобождение динамической памяти для одномерного массива выпол­няется с помощью операции

**delete [ ] А**, где **А** – имя массива, память под который выделялась операцией new.

1. Для освобождения динамической памяти для двухмерного массива нужно использовать цикл.

### Рекомендации по созданию программы

1. Определить состав и типы исходных данных, результатов и промежуточных величин.
2. Записать алгоритм сначала в общем виде, стремясь разбить его на простую последовательность шагов, а затем детализировать каждый шаг.
3. Написать программу. При написании программы рекомендуется:

- давать переменным понятные имена;

- не пренебрегать содержательными комментариями;

- использовать промежуточную печать вычисляемых величин в удобном формате;

- при написании вложенных циклов и операторов ветвления следить за отступами;

- операторы инициализации накапливаемых в цикле величин задавать непо­средственно перед циклом, в котором они вычисляются.

4. Параллельно с написанием программы задать тестовые примеры, которые проверяют все ветви алгоритма и возможные диапазоны значений исход­ных данных. Исходные данные удобнее формировать в файле (по крайней мере, при отладке), не забывая проверять в программе успешность его от­крытия.

## Содержание отчета

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

- Постановка задачи;

- Разработка алгоритма;

- Кодирование (соответствующий раздел отчета называется «Текст программы»);

- Тестирование (соответствующий раздел отчета называется «Анализ результатов»).

Раздел «Разработка алгоритма» должен содержать схему алгоритма функции печати матрицы.

В разделе «Текст программы» должен быть приведен листинг программы, включающий необходимые комментарии.

## Контрольные вопросы

1. Как передать в функцию матрицу?
2. Как создать динамическую матрицу?
3. Как и когда нужно освобождать память, занимаемую динамической матрицей?
4. Как можно переставить строки динамической матрицы?
5. Пусть для матрицы double В[10][10]

cout<<B; //на экран вывелось 0af8

Что напечатают следующие операторы?

cout<<B+2;//???

cout<< B[0]+1 ;//???

1. Напишите фрагмент программы для чтения данных из файла в двухмерный массив, используя потоковый ввод.
2. Напишите фрагмент программы для записи данных из двухмерного массива в файл, используя потоковый вывод.

## Рекомендуемые источники информации

# Лабораторная работа 8 Шифрование текстовых файлов

## Цель работы

Использование потокового ввода – вывода для работы с текстовыми файлами.

Использование библиотечных функций для обработки символьных строк.

Шифрование символьных данных с переменным ключом.

Анализ результатов шифрования текстовых файлов.

## Задание

### 10.2.1 Разработка программы для шифрования текста

Разработать программу для кодирования и декодирования текста, хранящегося в файле, основанную на методе Цезаря[[1]](#footnote-1), но более «изощренную»: ключ, используемый для кодирования символа, будет зависеть от позиции символа в исходном тексте. Для того, чтобы кодирование и декодирование можно было бы выполнять одним и тем же способом, вместо сложения кода символа с постоянным ключом (как у Цезаря) для переменных ключей используйте операцию «исключающее ИЛИ»: <*код символа>* **^** <*ключ>*.

Длина ключа – 1 байт.

Для получения ключей используется так называемый «кодовый блокнот», который представляет собой некоторый текст, известный только отправителю и получателю.

Количество символов в тексте не изменяется, а для кодирования и декодирования одинаковых по порядку символов в исходном и зашифрованном тексте должны использоваться одинаковые ключи.

Ключи представляют собой целые числа от 0 до 255, определяемые путем преобразования слов кодового блокнота по следующему алгоритму:

* Ключ для k-го по порядку символа в исходном тексте вычисляется как сумма (по модулю 256) кодов символов k-го по порядку слова кодового блокнота.
* Если кодовый блокнот имеет слов меньше, чем количество символов в исходном тексте, то по исчерпании слов в нём перейти к первому слову и продолжить кодирование.

Целесообразно сначала на основе кодового блокнота сформировать по заданному правилу целочисленный массив ключей, который затем будет использоваться при шифровании. ***Получение массива ключей оформить в виде отдельной функции и протестировать ее.***

### 10.2.2 Анализ результатов кодирования

Проанализировать зависимость статистики кодирования исходного текста (сколько каких кодов использовано для представления одного и того же исходного символа) от размера кодового блокнота для заданного исходного текста. Чем больше кодов используется для кодирования одного и того же символа, тем сложнее распознать текст, не зная алгоритма кодирования и текста кодового блокнота. Для наглядности результатов данные по кодированию выбранного символа исходного текста представить в виде таблицы 16х16, по аналогии с кодировочными таблицами. Исследование и вывод таблиц результатов следует выполнять в режиме диалога, последовательно вычисляя и выводя результаты для одного из двух запрашиваемых символов: пробела и какого-либо другого символа исходного текста для двух кодовых блокнотов различной длины, например, 0,5 Кбайт и 5 Кбайт. Для большей достоверности статистических результатов в качестве исходного текста использовать файлы размером не менее 20 Кбайт.

Планируемое время выполнения работы - 8 часов.

*Указание.*

В программе статистические данные собирать в массиве int stat[256], из которого они распечатываются в виде таблицы 16х16. (Одна таблица для одного символа исходного текста, код которого должен вводиться с клавиатуры по запросу.)

Индексы элементов массива stat[] представляют собой коды символов в зашифрованном тексте, в которые в результате кодирования преобразуется выбранный для сбора статистики символ исходного текста, а значения элементов массива stat[] равны числу повторных использований этого кода. (Один и тот же исходный символ в закодированном тексте представляется различными кодами в зашифрованном тексте и каждый код может повторятся несколько раз. Например, если используется только два ключа, то один и тот же символ исходного текста может быть представлен только двумя кодами и эти коды не обязательно будут печатными символами).

Для шифрования файла нет необходимости хранить весь текстовый файл в оперативной памяти.

При формировании числового массива ключей следует организовать пословное чтение файла с использованием функций getline() и strtok\_s(), а при шифровании и сборе статистики – посимвольное с помощью функций двоичного чтения данных get(ch) и put(ch). При открытии файлов устанавливать режим ios::binary. Использование бинарного режима для открытия текстового файла позволит исключить непредсказуемое влияние на процесс чтения-записи файла управляющих символов, коды которых могут появиться в тексте в результате шифрования.

## Рекомендации по выполнению работы

!!! Работа состоит из трех частей и рассчитана на 12 часов:

подготовительная часть - 4 часа (внеаудиторная самостоятельная работа),

основная часть – 8 часов аудиторных занятий (2 занятия по 4 часа).

При подготовке к ЛР выполните приведенные в пунктах 3.1 **÷** 3.5 примеры программ работы с символьными строками и файлами. ***Отладку программ для выполнения примеров выполняйте по шагам***, как это делается в примерах. Особое внимание обратите на последовательность шагов разработки программы и на применение библиотечных функций.

Для изучения функций используйте отладчик. Запомните: затраты времени на изучение функций библиотеки языка программирования, имеющих отношение к решаемой задаче, окупятся при разработке программы.

Первое занятие должно завершиться тестированием кодирования и декодирования различных текстовых файлов.

На втором занятии выполняется сбор и вывод статистических данных по шифрованию.

Для удобства отладки, выполнения кодирования и декодирования текстовых файлов и демонстрации результатов функция main() ***должна содержать меню***, позволяющее выполнять следующие действия:

- создание из кодового блокнота массива ключей,

- просмотр массива ключей,

- кодирование текстового файла,

- декодирование текста из файла с зашифрованным текстом,

- сбор статистики по результатам кодирования символов.

### Ввод-вывод строк

Для ввода-вывода строк используются как уже известные нам объекты cin и cout, так и функции, унаследованные из библиотеки С. Рассмотрим сначала первый способ:

#include <iostream.h>

int main()

{

const int n = 80;

char s[n];

cin >> s; cout << s << endl;

return ();

}

Как видите, строка вводится точно так же, как и переменные известных нам типов. Запустите программу и введите строку, состоящую из одного слова. Запустите про­грамму повторно и введите строку из нескольких слов. Во втором случае выводится только первое слово. Это связано с тем, что ввод выполняется до первого пробельно­го символа (то есть пробела, знака табуляции или символа перевода строки '\n')[[2]](#footnote-2). Можно ввести слова входной строки в отдельные строковые переменные:

#include <iostream>

int main()

{

const int n = 80;

char s[n], t[n], r[n];

cin >>s >>t >> r; cout <<s << endl << t << endl << r << endl**;**

return 0;

}

Если требуется ввести строку, состоящую из нескольких слов, в одну строковую переменную, используются *методы* getline или get класса istream, объектом кото­рого является cin. Во втором семестре мы изучим, что такое методы клас­са, а пока можно пользоваться ими, не вдумываясь в смысл. Единственное, что нам пока нужно знать, это синтаксис вызова метода — после имени объекта ставится точка, а затем пишется имя метода:

#include <iostream>

int main()

{

const int n = 80;

char s[n];

cin.getline(s, n);

cout << s << endl;

cin.get(s, n);

cout << s << endl;

return 0;

}

Метод getline считывает из входного потока n - 1 символов или менее (если сим­вол перевода строки встретится раньше) и записывает их в строковую перемен­ную s. Символ перевода строки[[3]](#footnote-3) также считывается (удаляется) из входного потока, но не записывается в строковую переменную, вместо него размещается заверша­ющий 0. Если в строке исходных данных более n-1 символов, следующий ввод будет выполняться из той же строки, начиная с первого несчитанного символа. Метод get с двумя аргументами работает аналогично, но оставляет в потоке символ перевода строки. В строковую переменную добавляется завершающий 0.

Никогда не обращайтесь к разновидности метода get с двумя аргументами два раза подряд, не удалив \n из входного потока. Например,

cin.get(s, n); // *1-* считывание строки

cout << s << endl; // *2-* вывод строки

cin.get(s, n); // *3-* считывание строки

cout << s << endl; // *4-* вывод строки

cin.get(s, n); // *5 -* считывание строки

cout <<s << endl; // *6-* вывод строки

cout << "Конец - делу венец" <<endl; // 7

При выполнении этого фрагмента вы увидите на экране первую строку, выведен­ную оператором 2, а затем завершающее сообщение, выведенное оператором 7. Какие бы прекрасные строки вы ни ввели с клавиатуры в надежде, что они будут прочитаны операторами 3 и 5, метод get в данном случае «уткнется» в символ \n, оставленный во входном потоке от первого вызова этого метода (оператор 1). В ре­зультате будут считаны и, соответственно, выведены на экран пустые строки (стро­ки, содержащие 0 символов). А символ \n так и останется «торчать» во входном потоке. Возможное решение этой проблемы — удалить символ \n из входного по­тока путем вызова метода char get(), который всегда извлекает символ из потока, то есть после операторов 1 и 3 нужно вставить вызов cin.get().

Однако есть и более простое решение — использовать в таких случаях метод getline, который после прочтения строки не оставляет во входном потоке сим­вол \n.

Если в программе требуется ввести несколько строк, метод getline удобно исполь­зовать в заголовке цикла, например:

#include <iostream>

int main()

{

const int n = 80;

char s[n];

while (cin.getline(s, n))

{

cout << s << endl;

… // обработка строки

}

return 0;

}

### Пример программы работы с символьными строками.

*Написать программу, которая определяет, сколько раз встретилось заданное слово в текстовом файле, длина строки в котором не превышает 80 символов. Текст не содержит переносов слов.*

Определим слово как последовательность алфавитно-цифровых символов, после которых следует знак пунктуации, разделитель или признак конца строки. Слово может находиться либо в начале строки, либо после разделителя или знака пунк­туации. Это можно записать следующим образом (фигурные скобки и вертикаль­ная черта означают выбор из альтернатив):

слово = {начало строки | знак пунктуации | разделитель} <символы, составляющие слово> {конец строки | знак пунктуации | разделитель}

**I. *Исходные данные и результаты***

Исходные данные:

1. Текстовый файл неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно ограни­читься поиском слова в каждой строке отдельно. Для ее хранения выделим строку длиной 81 символ.
2. Слово для поиска, вводимое с клавиатуры. Для его хранения также выделим строку длиной 81 символ.

Результатом работы программы является количество вхождений слова в текст. Представим его в программе в виде целой переменной.

Для хранения длины строки будем использовать именованную константу, а для хранения фактического количества символов в слове — переменную целого типа. Для работы с файлом будем использовать потоковый ввод-вывод.

**II. *Алгоритм решения задачи***

- Построчно считывать текст из файла.

- Просматривая каждую строку, искать в ней заданное слово. При каждом  
нахождении слова увеличивать счетчик.

Детализируем второй пункт алгоритма. Очевидно, что слово может встречаться в строке многократно, поэтому для поиска следует организовать цикл просмотра строки, который будет работать, пока происходит обнаружение в строке последо­вательности символов, составляющих слово.

При обнаружении совпадения с символами, составляющими слово, требуется определить, является ли оно отдельным словом, а не частью другого. (Это один из возможных вариантов решения задачи, не самый лучший. Другой вариант – сначала выделить слово, а затем сравнивать его с заданным). Например, мы задали слово «кот». Эта последовательность символов содержится, например, в словах «котенок», «трикотаж», «трескотня» и «апперкот». Следовательно, требуется проверить символ, стоящий после слова, а в случае, когда слово не находится в начале строки — еще и символ перед словом. Эти символы проверяются на принадлежность множеству знаков пунктуации и разделителей.

III. ***Программа и тестовые примеры***

Разобьем написание программы на последовательность шагов.

*Шаг 1.* Ввести «скелет» программы (директивы #include, функцию main(), описа­ние переменных, открытие файла). Добавить контрольный вывод введенного сло­ва. Запустив программу, проверить ввод слова и успешность открытия файла. Исправить ошибки в коде, если они имеются. Для проверки вывода сообщения об ошибке следует выполнить программу еще раз, задав имя несуществующего файла.

#include <fstream >

using namespace std;

int main()

{

const int len = 81;

char word[len],line[len];

cout << " Input the word for search: ";

cin >> word;

ifstream fin("text.txt",ios::in);

if (!fin) { cout << "Error of file opening."<< endl;

return 1; }

return 0;

}

*Шаг 2.* Добавить в программу цикл чтения из файла, внутри цикла поставить кон­трольный вывод считанной строки:

#include <fstream>

int main()

{

const int len = 81;

char word[len], line[len];

cout << "Input the word for search: ";

cin >> word;

ifstream fin("text.txt", ios::in);

if (!fin) { cout << "Error of file opening."<< endl;

return 1;}

while (fin.getline(line, len))

cout << line << endl;

return 0;

}

*Шаг З.* Добавить в программу цикл поиска последовательности символов, состав­ляющих слово, с контрольным выводом:

#include <fstream>

int main()

{

const int len = 81;

char word[len], line[len];

cout << "Input the word for search: ";

cin >>word;

int l\_word = strlen(word);

ifstream fin("text.txt", ios::in);

if (!fin) { cout << "Error of file opening."<< endl;

return 1;

}

int count =0;

while (fin.getline(line, len)) {

char \*p = line;

while( p = strstr(p, word)) {

cout << “совпадение: " << p << endl;

p += l\_word;

count++;

}

}

cout << count << endl;

return 0;

}

Для многократного поиска вхождения подстроки в заголовке цикла используется функция strstr. Очередной поиск должен выполняться с позиции, следующей за найденной на предыдущем проходе подстрокой. Для хранения этой позиции определяется вспомогательный указатель р, который на каждой итерации цикла наращивается на длину подстроки. Также вводится счетчик количества совпадений. На данном этапе он считает не количество слов, а количество вхождений последовательности символов, составляющих слово.

*Шаг 4.* Добавить в программу анализ принадлежности символов, находящихся перед словом и после него, множеству знаков пунктуации и разделителей:

#include <fstream>

#include <ctype.h>

int main()

{

const int len = 81;

char word[len], line[len];

cout << "Input the word for search: ";

cin >>word;

int l\_word = strlen(word);

ifstream fin("text.txt", ios::in);

if (!fin) { cout << "Error of file opening."<< endl;

return 1;

}

int count =0;

while (fin.getline(line, len)) {

char \*p = line;

while( p = strstr(p, word)) {

char \*c = p; //с-начало подстроки совпадения

p += l\_word; //р-конец подстроки совпадения

// подстрока не в начале строки?

if (c != line)

// символ перед подстрокой совпадения не разделитель?

if ( !ispunct(\*(c - 1) ) && !isspace(\*(c - 1) )) continue;

// символ после слова разделитель?

if ( ispunct(\*p) || isspace(\*p) || (\*p == '\0') ) count++; //подстрока – отдельное слово

}

}

cout << "Number of entering word: "<< count << endl;

return 0;

}

Здесь вводится служебная переменная ***c*** для хранения адреса начала вхождения подстроки. Символы, ограничивающие слово, проверяются с помощью функций ispunct и isspace, прототипы которых хранятся в заголовочном файле <ctype.h>. Символ, стоящий после слова, проверяется также на признак конца строки (для случая, когда искомое слово находится в конце строки).

Для тестирования программы требуется создать файл с текстом, в котором задан­ное слово встречается:

- в начале строки;

- в конце строки;

- в середине строки;

- несколько раз в одной строке;

- как часть других слов, находящаяся в начале, середине и конце этих слов;

- в скобках, кавычках и других разделителях.

Длина хотя бы одной из строк должна быть равна 80 символам. Для тестирования программы следует выполнить ее по крайней мере два раза: введя с клавиатуры слово, содержащееся в файле, и слово, которого в нем нет.

***Давайте теперь рассмотрим другой вариант решения этой задачи***. В библиотеке есть функция strtok\_s, которая разбивает переданную ей строку на лексемы в соот­ветствии с заданным набором разделителей. Если мы воспользуемся этой функ­цией, нам не придется «вручную» выделять и проверять начало и конец слова, потребуется лишь сравнить с искомым словом слово, выделенное с помощью strtok\_s. Правда, список разделителей придется задать вручную.

Прототип функции:

char\* strtok\_s(char\* token, const char\* delims, char\*\*next\_token);

Здесь:

char\* token –адрес выбираемого из строки слова   
 char\*\*next\_token – адрес следующего слова в строке или 0 в конце строки  
 char\* delims – строка разделителей

#include <fstream>

#include <iostream>

#include <string>

using namespace std;

int main()

{

const int len = 1000;

char word[80], line[len];

char \*delims = ".,!? /<>|)(\*::\"";

cout << "Input the word for search: "; cin >> word;

ifstream fin("text.txt", ios::in);

if (!fin) {

cout << "Error of file opening."<< endl;

system("pause");

return 1;

}

char \*token,\*next\_token;

int count = 0;

while (fin.getline(line, len)) {

next\_token = line;

while( token = strtok\_s(next\_token,delims,&next\_token) ) { // 1

if ( !strcmp (token, word) ) count++; // 2

}

}

cout << "Number of entering word: "<<count << endl;

system("pause");

return 0;

}

Первый вызов функции strtok\_s в операторе 1 возвращает адрес первой лексемы (сло­ва) строки line. Он сохраняется в переменной token. Функция strtok\_s заменяет на NULL разделитель, находящийся после найденного слова, поэтому в операторе 2 можно сравнить на равенство искомое и выделенное слово.   
&next\_token – адрес следующей лексемы в той же строке.

Как видите, программа стала короче и яснее. На этом примере можно видеть, что средства, предоставляемые языком, влияют на алгоритм решения задачи, и поэто­му перед тем, как продумывать алгоритм, необходимо эти средства изучить. Пред­ставьте, во что бы вылилась программа без использования функций работы со стро­ками и символами!

### Работа с файлами

Информация во внешней памяти (на диске, на магнитных лентах и т.п.) сохраняется в виде файлов - именованных объектов, доступ к которым обеспечивает операционная система ЭВМ. Основное отличие внешней памяти ЭВМ от основной (иначе опе­ративной) памяти - возможность сохранения информации при отклю­чении ЭВМ. Поддержка операционной системы состоит в том, что в ней имеются средства:

1. создания файлов;
2. уничтожения файлов;
3. поиска файлов на внешнем носителе информации (на диске);
4. чтения и записи данных из файлов и в файлы;
5. открытия файлов;
6. закрытия файлов;
7. позиционирования файлов.

Библиотека ввода-вывода Си++ включает средства для работы с последовательными файлами. Логически последовательный файл можно представить как именованную цепочку (ленту, строку) байтов, имеющую начало и конец. Чтение (или запись) из файла (в файл) ведутся байт за байтом от начала к концу. В каждый момент позиции в файле, откуда выполняется чтение и куда производится запись, определяются значениями указателей по­зиций записи и чтения файла. Позиционирование указателей записи и чтения (т.е. установка на нужные байты) выполняется либо автомати­чески, либо за счет явного управления их положением. В стандартной библиотеке ввода-вывода Си++ имеются соответствующие средства.

Взаимосвязь файлов с потоками ввода-вывода осуществляется с помощью следующих действий:

1. - создание файла;
2. - создание потока;
3. - открытие файла;
4. - "присоединение" файла к потоку;
5. - обмены с файлом с помощью потока;
6. - "отсоединение" потока от файла;
7. - закрытие файла;
8. - уничтожение файла.

Все перечисленные действия могут быть выполнены с помощью средств библиотеки классов ввода-вывода языка Си++. Однако существует несколько альтернативных вариантов их выполнения.

Потоки для работы с файлами создаются как объекты следующих классов:

1. ofstream - для вывода (записи) данных в файл;
2. ifstream - для ввода (чтения) данных из файла;
3. fstream -для чтения и для записи данных (двунаправленный обмен).

Чтобы использовать эти классы, в текст программы необходимо включить дополнительный заголовочный файл fstream. После этого в программе можно определять конкретные файловые потоки, соответст­вующих типов (объекты классов ofstream, ifstream, fstream), например, таким образом:

**ofstream** outFile; // Определяется выходной файловый поток

**ifstream** inFile; // Определяется входной файловый поток

f**stream** ioFile; // Определяется файловый поток для ввода и вывода

Создание файлового потока (объекта соответствующего класса) связывает имя потока с выделяемым для него буфером и инициализи­рует переменные состояния потока. Так как перечисленные классы файловых потоков наследуют свойства класса ios, то и переменные состояния каждого файлового потока наследуются из этого базового класса. Так как файловые классы являются производными от классов ostream (класс **ofstream),** istream (класс ifstream), stream (класс **fstream),** то они поддерживают описанный ниже форматированный и бесформатный обмен с файлами для этих классов. Однако прежде чем выполнить обмен, необходимо открыть соответствующий файл и связать его с файловым потоком.

Открытие файла в самом общем смысле означает процедуру, ин­формирующую систему о тех действиях, которые предполагается вы­полнять с файлом. Для работы с файло­выми потоками библиотеки ввода-вывода языка Си++ удобно пользоваться компонентными функциями (методами) соответствующих классов.

Создав файловый поток, можно "присоединить" его к конкретно­му файлу с помощью компонентной функции **ореn**(). Функция **open**() унаследована каждым из файловых классов **ofstream, ifsream, fstream** от класса f**streambase**. С ее помощью можно не только открыть файл, но и свя­зать его с уже определенным потоком. Формат функции:

void open(const char \*fileName, int mode = умалчиваемое\_значение,

int protection = умалчиваемое\_значение);

Первый параметр - fileName - имя уже существующего или соз­даваемого заново файла. Это строка, определяющая полное или со­кращенное имя файла в формате, регламентированном операционной системой. Второй параметр - mode *(режим)* - дизъюнкция флагов, определяющих режим работы с открываемым файлом (например, только запись или только чтение). Флаги определены следующим об­разом:

enum ios::open\_mode {

**in** = 0x01, // Открыть только для чтения

**out** = 0x02, // Открыть только для записи

**арр =** 0x08**, //** Дописывать данные в конец файла

**trunc =** 0x10**, //** Вместо существующего создать новый файл

**binary =** 0x80**, //** Открыть для двоичного (не текстового) обмена

}**;**

Назначения флагов поясняют комментарии, однако надеяться, что именно такое действие на поток будет оказывать тот или иной флаг в конкретной реализации библиотеки ввода-вывода, нельзя. Как пишет автор языка Си++ [26], "смысл значений open\_mode скорее всего зави­сит от реализации". Умалчиваемое значение параметра mode зависит от типа пото­ка, для которого вызывается функция open ().

Третий параметр - protection *(защита) -* определяет защиту и достаточно редко используется. Точнее, он устанавливается по умол­чанию и умалчиваемое значение обычно устраивает программиста.

Как обычно вызов функции open () осуществляется с помощью уточненного имени

*имя\_объекта\_класса.вызов\_принадлежащей\_классу\_функции*

Итак, открытие и присоединение файла к конкретному файловому потоку обеспечивается таким вызовом функции open ():

*имя потока.open(имя\_файла, режим, защита);*

Здесь имя\_потока - имя одного из объектов, принадлежащих классам ofstream, ifstream, fstream. Примеры вызовов для опреде­ленных выше потоков:

*outFile.open("С:\\USER\\RESULT.DAT");*

*inFile.open("DATA.TXT");*

*ioFile.open("CHANGE.DAT",ios::out);*

При открытии файлов с потоками класса **ofstream** второй параметр по умолчанию устанавливается равным **ios:: out**, т.е. файл открывается только для вывода. Таким образом, файл **с: \user\resalt.dat** после удачного выполнения функции **open ()** будет при необходимости (если он не существовал ранее) создан, а затем открыт для вывода (записи) данных в текстовом режиме обмена и присоединен к потоку **outFile**. Теперь к потоку **outFile** может применяться, например, операция включения <<, как к стандартным выходным потокам **cout, cerr**.

Поток **inFile** класса **ifstream** в нашем примере присоединяется функцией **open()** к файлу с именем **data.txt**. Этот файл открывается для чтения из него данных в текстовом режиме. Если файла с именем **data.txt** не существует, то попытка вызвать функцию **inFile.open ()** приведет к ошибке.

Для проверки удачности завершения функции open () используется перегруженная операция **!**. Если унарная операция ! применяется к потоку, то результат ненулевой при наличии ошибок. Если ошибок не было, то выражение **!имя\_потока** имеет нулевое значение. Таким образом, можно проверить результат выполнения функции **open():**

if (!inFile)// лучше для проверки использовать метод fail(): if(inFile.fail())

{ cerr « "Ошибка при открытии файла!\n";

exit(1);

Для потоков класса **fstream** второй аргумент функции **open()** должен быть задан явно, так как по умолчанию неясно, в каком направлении предполагается выполнять обмен с потоком. В примере файл **change.dat** открывается для записи и связывается с потоком **ioFile**, который будет выходным потоком до тех пор, пока с помощью повторного открытия файла явно не изменится направление обмена с файлом или потоком.

В классах **ifstream, ofstream, fstream** определены конструкто­ры, позволяющие по-иному выполнять создание и открытие файлов. Типы конструкторов для потоков разных классов очень похожи:

**имя\_класса()** ;

создает поток, не присоединяя его ни к какому файлу;

**имя\_класса(int fd);**

создает поток и присоединяет его к уже открытому файлу, де­скриптор которого используется в качестве параметра **fd**;

**имя\_класса(int fd, char \*buf, int);**

создает поток, присоединяя его к уже открытому файлу с де­скриптором **fd**, и использует явно заданный буфер (параметр buf);

**имя\_класса(char \*FileName, int mode, int = ...);**

создает поток, присоединяет его к файлу с заданным именем Filename, а при необходимости предварительно создает файл с таким именем.

Детали и особенности перечисленных конструкторов лучше изучать по документации конкретной библиотеки ввода-вывода.

Работая со средствами библиотечных классов ввода-вывода, чаще всего употребляют конструктор без параметров и конструктор, в котором явно задано имя файла. Примеры обращений к конструкторам без параметров:

**ifstream fi;** //Создает входной файловый поток fi

**ostream fo;** // Создает выходной файловый поток fo

**fstream ff;** // Создает файловый поток ввода-вывода ff

После выполнения каждого из этих конструкторов файловый поток можно присоединить к конкретному файлу, используя уже упомянутую компонентную функцию **open ()** :

**void open (char\*FileName, int режим, int защита);**

Примеры:

fi.open ("Filel.txt",ios::in); // Поток fi соединен с файлом Filel.txt

fi.close(); //Разорвана связь потока fi с файлом Filel.txt

fi.open("File2.txt"); //Поток fi присоединен к файлу File2.txt

fo.open("NewFile"); // Поток fo присоединяется к файлу NewFile; если такой файл //отсутствует - он будет создан

При обращении к конструктору с явным указанием в параметре имени файла остальные параметры можно не указывать, они выби­раются по умолчанию.

Примеры:

**ifstream flow1 ("File.1");**

создает входной файловый поток с именем **flow1** для чтения данных. Разыскивается файл с названием **File.1**. Если такой файл не существует, то конструктор завершает работу аварийно. Проверка:

**if (!flow1) cerr << "Не открыт файл File.1!";**

**ofstream flow2 ("File.2");**

создается выходной файловый поток с именем **flow2** для за­писи информации. Если файл с названием **File.2** не су­ществует, он будет создан, открыт и соединен с потоком **f low2.** Если файл уже существует, то предыдущий вариант бу­дет удален и пустой файл создается заново. Проверка:

**if (!flow2) cerr << "Не открыт файл File.2!";**

**fstream flow3("File.3");**

создается файловый поток **fiow3**, открывается файл **File.3** и присоединяется к потоку **flow3**.

Все файловые классы унаследовали от базовых классов функцию **close** (), позволяющую очистить буфер потока, отсоединить поток от файла и закрыть файл. Функцию **close** () необходимо явно вызывать при изменении режимов работы с файловым потоком. Автоматически эта функция вызывается только при завершении программы.

В качестве иллюстрации основных особенностей работы с файла­ми рассмотрим несколько программ.

//Чтение текстового файла с помощью onepaции >>

#include <stdlib.h> // Для функции exit()

#include <fstream > // Для файловых потоков

const int lenName = 13; // max длина имени файла

const int lenString = 60; // Длина вспомогательного массива

void main()

{

char source[lenName]; // Массив для имени файла

cout << "\nВведите имя исходного файла: ";

cin >> source;

ifstream inFile; // Входной файловый поток

// Открыть файл source и связать его с потоком inFile:

inFile.open(source);

if (!inFile) // Проверить правильность открытия файла

{ cerr « "\nОшибка при открытии файла " << source;

exit(1); // Завершение программы

}

// Вспомогательный массив для чтения:

char string[lenString];

char next;

cout << "\n Текст файла:\n\n";

cin.get(); // Убирает код из потока cin

while(1) // Неограниченный цикл

{ // Ввод из файла одного слова до пробельного символа либо EOF:

inFile >> string;

// Проверка следующего символа:

next = inFile.peek();

// Выход при достижении конца файла:

if (next == EOF) break;

// Печать с добавлением разделительного пробела:

cout << string <<” “;

if (next == '\n') // Обработка конца строки

{ cout << '\n';

// 4 - смещение для первой страницы экрана:

static int i = 4;

// Деление по страницам до 20 строк каждая:

if (!(++i % 20))

{ cout << "\nДля продолжения вывода нажмите ENTER.\n" << endl;

cin.get() ;

}

}

}

Результат выполнения программы - постраничный вывод на эк­ран текстового файла, имя которого набирает на клавиатуре пользо­ватель по "запросу" программы. Размер страницы - 20 строк. В начале первой страницы - результат диалога с пользователем и поэтому из файла читаются и выводятся только первые 16 строк.

Программа демонстрирует неудобства чтения текста из файла с помощью операции извлечения >>, которая реагирует на каждый об­общенный пробельный символ. Между словами, прочитанными из файла, принудительно добавлено по одному пробелу. А сколько их (пробелов) было в исходном тексте, уже не известно. Тем самым иска­жается содержащийся в файле текст. Читать пробельные символы по­зволяет компонентная функция **getline()** класса **istream**, наследуемая классом **ifstream**. Текст из файла будет читаться и вы­водиться на экран (в поток **cout**) без искажений (без пропусков про­белов), если в предыдущей программе чтение и вывод в поток **cout** организовать таким образом:

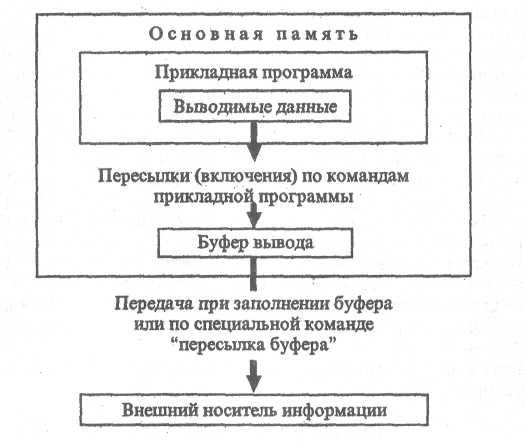
while( inFile.getline(string,lenString)

cout << string;

…..

### Потоки ввода-вывода.

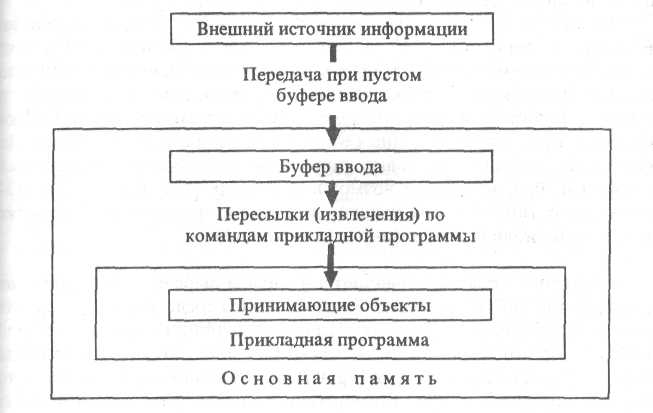
Всоответствии с названием заголовочного файла **iostream (stream** - поток; "i" - сокращение от *input - ввод* "o" - сокращение от *output -* вывод) описанные в этом файле средства ввода-вывода обеспечивают программиста механизмами для извлечение данных из потоков и для включения (внесения) данных в потоки. Поток определяется как последовательность байтов (символов) и с точки зрения программы не зависит от тех конкретных устройств (файл на диске, принтер, клавиатура, дисплей, стример и т.п.), с которыми ведется обмен данными. При обмене с потоком часто используется вспомогательный участок основной памяти - буфер потока (рис.1**-** буфер вывода, рис. **2** - буфер ввода).



***Рис.*** *1. Буферизированный выходной поток*

В буфер потока помещаются выводимые программой данные перед тем, как они будут переданы к внешнему устройству. При вводе данных они вначале помещаются в буфер и только затем передаются в область памяти выполняемой программы. Использование буфера как промежуточной ступени при обменах с внешними устройствами по­вышает скорость передачи данных, так как реальные пересылки осу­ществляются только тогда, когда буфер уже заполнен (при выводе) или пуст (при вводе).

Работу, связанную с заполнением и очисткой буферов ввода-вывода, операционная система обычно берет на себя и выполняет без явного участия программиста. Поэтому поток в прикладной про­грамме можно рассматривать просто как последовательность байтов. При этом очень важно, что никакой связи значений этих байтов с кодами какого-либо алфавита не предусматривается. Задача программиста при вводе-выводе с помощью потоков - установить со­ответствие между участвующими в обмене типизированными объек­тами и последовательностью байтов потока, в которой отсутствуют всякие сведения о типах представляемой (передаваемой) информации.



***Рис .2****. Буферизированный входной поток*

Используемые в программах потоки логически делятся на три типа:

1. входные, из которых читается информация;
2. выходные, в которые вводятся данные;
3. двунаправленные, допускающие как чтение, так и запись.

Все потоки библиотеки ввода-вывода последовательные, т.е. в каждый момент для потока определены позиции записи и (или) чтения, и эти позиции после обмена перемещаются по потоку на длину переданной порции данных.

#### Функции для обмена с потоками

Кроме операции включения (записи) в поток << и извлечения (чтения) из потока >>, в классах библиотеки ввода-вывода есть весьма полезные функции, обеспечивающие программиста альтернативными средствами для обмена с потоками.

При выводе в качестве основного класса, формирующего выход­ные потоки, используется класс **ostream.** В нем определены (ему при­надлежат) две функции для двоичного вывода данных:

**ostream& ostream::put(char cc);**

**ostream& ostream::write(const signed char \*array, int n);**

**ostream& ostream::write(const unsigned char \*array, int n) ;**

Функция **put ()** помещает в тот выходной поток, для которого она вызвана, символ, использованный в качестве фактического парамет­ра.

В этом случае эквивалентны операторы:

**cout << 'Z';**

и

**cout.put('Z');**

Функция **write ()** имеет два параметра - указатель **array** на учас­ток памяти, из которого выполняется вывод, и целое значение **n**, определяющее количество выводимых из этого участка символов (байт).

В отличие от операции **<<** включения в поток функции **put ()** и **write** () не обеспечивают форматирования выводимых данных. На­пример, если при выводе одного символа с помощью операции << можно, используя функцию **width (),** разместить его в поле из нужно­го количества позиций, то функция **put()** всегда разместит символ в одной позиции выходного потока. Флаги форматирования также не применимы к функциям **put()** и **write().**

Так как функции **put()** и **write**() возвращают ссылки на объект того класса, для которого они выполняются, то можно организовать цепочку вызовов:

char ss[] = "Merci";

cout.put('\n').write(ss,sizeof(ss)-l).put('!').put('\n');

На экране (в потоке **cout**) появится:

**Merci!**

Если необходимо прочитать из входного потока строку символов, содержащую пробелы, то с помощью операции извлечения >> это де­лать неудобно - каждое чтение строки выполняется до пробела, а ве­дущие (левые) пробельные символы игнорируются. Если мы хотим, набрав на клавиатуре строку: "Qui vivra verra- будущее покажет (лат.) ", ввести ее в символьный массив, то с помощью операции извлечения >> это сделать несколько хлопотно, все слова будут читаться отдельно (до пробела). Гораздо удобнее воспользоваться функциями бесформатного (двоичного) чтения.

Функции двоичного (бесформатного) чтения данных принадлежат потоку **istream**. Прежде чем перечислить их, отметим основное свой­ство двоичного чтения данных. Данные читаются без преобразования их из двоичного представления в текстовое. Например, если во вход­ном потоке размещено представление вещественного числа

**1.3е-3**, то это будет воспринято как последовательность из шести байт, и читать эту последовательность с помощью функций двоичного ввода можно только в символьный массив.

#### Функции чтения

Во-первых, это 6 перегруженных функций get (). Две из них имеют следующие прототипы:

**istream& get(signed char \*array, int max\_len, char ='\n');**

**istream&** **get(unsigned char \*array, int max\_len, char ='\n');**

Каждая из этих функций выполняет извлечение (чтение) последо­вательности байтов из стандартного входного потока и перенос их в символьный массив, задаваемый первым параметром. Второй пара­метр определяет максимально допустимое количество прочитанных байтов. Третий параметр определяет ограничивающий символ (байт), при появлении которого во входном потоке следует завершить чтение. По умолчанию третий параметр имеет значение '\n' - переход на следующую строку, однако при обращении к функции его можно за­давать и по-другому. Значение этого третьего параметра из входного потока не удаляется, он в формируемую строку (символьный массив) не переносится, а вместо него автоматически добавляется "концевой" символ строки ' \0'. Если из входного потока извлечены ровно **max\_len - 1** символов, однако ограничивающий символ (например, по умолчанию '\n') не встретился, то концевой символ помещается после введенных символов. Массив, в который выполняется чтение, должен иметь длину не менее **max\_len** символов. Если из входного потока не извлечено ни одного символа, то устанавливается код ошибки. Если до появления ограничивающего символа и до извлече­ния **max\_len - 1** символов встретился конец файла **EOF**, то чтение прекращается как при появлении ограничивающего символа.

Функция с прототипом istream& get(streambuf& buf, char = '\n');

извлекает из входного потока символы и помещает их в буфер, опре­деленный первым параметром. Чтение продолжается до появления ограничивающего символа, которым по умолчанию является **'\n'**, но он может быть установлен явно любым образом.

Три следующих варианта функции **get()** позволяют прочесть из входного потока один символ. Функции

istream& get(char& cc) ;

присваивают извлеченный символ фактическому параметру и воз­вращают ссылку на поток, из которого выполнено чтение.

!!! ***Функция get(char& ) не извлекает из потока символ конца строки ‘\n’ (код 10).*** Чтобы исключить зацикливание программы, нужно перед вводом выполнять cin.ignore().

Функция

int get() ;

получает код извлеченного из потока символа в качестве возвращае­мого значения. Если поток пуст, то возвращается код конца файла **EOF**.

Функции "ввода строк":

istream& getline(signed char \*array, int len, char= '\n');

istream& getline(unsigned char \*array, int len, char= '\n');

подобны функциям **get ()** с теми же сигнатурами, но переносят из входного потока и символ-ограничитель. Функция

int peek () ;

позволяет "взглянуть" на очередной символ входного потока. Точнее, она возвращает код следующего символа потока (или **EOF**, если поток пуст), но оставляет этот символ во входном потоке. При необходимости этот символ можно в дальнейшем извлечь из потока с помощью других средств библиотеки. Например, следующий цикл работает до конца строки (до сигнала от клавиши **Enter ):**

**char cim;**

**while (cin.peek() != '\n')**

**{ cin.get(cim);**

**cout.put(cim) ; }**

Принадлежащая классу **istream** функция

**istream& putback (char cc)**;

не извлекает ничего из потока, а помещает в него символ **cc,** которыйстановится текущим и будет следующим извлекаемым из потока сим­волом.

Аналогичным образом функция

**int gcount ();**

подсчитывает количество символов, которые были извлечены из входного потока при последнем обращении к нему. Функция

**istream& ignore(int n=1, int EOF);**

позволяет извлечь из потока и "опустить" то количество символов **n**, которое определяется первым параметром. Второй параметр определяет символ-ограничитель, при появлении которого выполнение функции нужно прекратить, даже если из потока еще не извлечены все **n** символов. Функции

**istream& read(signed char \*array, int numb);**

**istream& read(unsigned char \*array, int numb);**

выполняют чтение заданного количества **numb** символов в массив **array.**

Полезны следующие функции того же класса **istream**:

**istream& seekg(long pos) ;**

устанавливает позицию чтения из потока в положение, опре­деляемое значением параметра.

**istream& seekg(long pos, seek\_dir dir);**

выполняет перемещение позиции чтения вдоль потока в на­правлении, определенном параметром **dir**, принимающим значения из перечисления **enum seek\_dir {beg, cur, end }.** Относительная величина перемещения (в байтах) определяется значением параметра **long pos**. Если направление определено как **beg**, то смещение от начала потока; **cur** - от текущей позиции; **end** - от конца потока;

**long tellg()**

определяет текущую позицию чтения из потока.

Подобные перечисленным функции класса **ostream**:

**long tellp()**

определяет текущую позицию записи в поток:

**ostream& seekp (long pos, seek\_dir dir)**

аналогична функции **seekg ()** , но принадлежит классу **ostream**

и выполняет относительное перемещение позиции записи

в поток;

**ostream& seekp(long pos);**

устанавливает абсолютную позицию записи в поток.

### Использование аргументов командной строки

Данные в функцию main() можно передавать через параметры из командной строки (строки запуска функции на выполнение из операционной системы). Это удобно при отладке программы (чтобы не вводить при каждом запуске одни и те же данные) и может быть удобным при использовании программы.

При использовании функции main() с параметрами в командной строке заголовок функции имеет вид:

int main(int argc, char\* argv[]), где:

argc – количество параметров;

argv[] – массив указателей на параметры (каждый параметр – строка).

Первый параметр, argv[0] – указатель на полное имя исполняемого файла. Он передается по умолчанию и не указывается при запуске программы.

Для выполнения программы с аргументами в командной строке нужно уметь:

- запустить программу с параметрами в командной строке;  
- проконтролировать из программы количество параметров;  
- использовать параметры из командной строки в тексте программы.

Запуск программы с параметрами в командной строке можно осуществить непосредственно из командной строки ДОС или из среды программирования Microsoft Visual C++.

Для запуска из Microsoft Visual C++ нужно:

- открыть меню **Project;**

**-** выбрать пункт **Settings…**;

**-** в окне **Program settings** выбрать закладку **Debug**;

**-** в поле **Program arguments** через пробел ввести необходимые параметры.

Для запуска из командной строки ДОС нужно выполнить следующие действия:

- на Рабочем столе Windows нажать кнопку ***ПУСК;***

- в раскрывшемся меню выбрать пункт ***ВЫПОЛНИТЬ;***

- с помощью кнопки ***ОБЗОР***… найти нужный файл с расширением .exe;

- в окне **Открыть:** после символа “, стоящего в конце полного имени файла, через пробел ввести необходимые параметры.

При контроле числа параметров нужно иметь ввиду, что первый параметр всегда передается по умолчанию и число параметров будет на 1 больше, чем введено. Например, если передается один параметр - имя файла с исходными данными, то argc будет равно 2, argv[0] указывает на полное имя исполняемого модуля (точнее, на список параметров системного окружения, первым из которых является имя исполняемого файла), а argv[1] указывает на введенное имя файла с данными.

При использовании параметров нужно помнить, что каждый передаваемый параметр – это символьная строка. Если надо передать численное значение, то в программе надо преобразовать строку в число с помощью соответствующей типу числа функции (double atof(char \*str), int atoi(char \*str), или long atol(char \*str)).

В качестве примера работы с файловыми потоками приведем программу копиро­вания одного файла в другой. Имена файлов берутся из аргументов командной строки:

// МуСору.срр

#include <iostream>

#include <fstream>

using namespace std;

int main(int argc, char\* argv[])

{

if (argc != 3) cout <<"Неверное число аргументов"<<endl;

ifstream from(argv[1]); // открываем входной файл

if (!from)

{

cout<<"Входной файл "<< argv[1 ]<<" не найден"<<endl;

return 1;

}

ofstream to(argv[2]); // открываем выходной файл

if (!to)

{

cout<<"Выходной файл "<< argv[2]<< " не открыт"<< endl;

return1;

}

char ch;

while (from.get(ch))

{

to.put(ch);

if (!to) cout<<"Ошибка записи (диск переполнен).";

}

cout << "Копирование из " << argv[1] << " в " << argv[2] << " завершено." << endl;

return 0;

}

## Содержание отчета

## Контрольные вопросы и задания

* 1. Создание текстовых файлов.
  2. Использование методов getline() и strtok\_s() для пословного чтения текста из файла.
  3. Создание бинарных файлов.
  4. Использование методов put(),write() и get(),read() для работы с бинарными файлами.
  5. Обменяйтесь зашифрованными письмами с двумя студентами, используя для переписки с каждым из них различные кодовые блокноты.

## Рекомендуемые источники информации

Основные.

1. Р. Лафоре. Объектно-ориентированное программирование в С++. Издательство ПИТЕР, 2004 г.  
2. Г. Шилдт. Полный справочник по С++, 4-е издание, Издательский дом «Вильямс», 2006Дополнительные.  
3. Н. Вирт. Алгоритмы+Структуры данных=Программы, «Мир», 1985 г.  
4. <http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.

# Лабораторная работа 9 Вычисление обратной матрицы методом Гаусса-Жордана

## Цель работы

Разработка функции, реализующей алгоритм вычисления обратной матрицы методом Гаусса-Жордана.

Разработка функции умножения матриц.

Использование разработанных функций для решения систем линейных уравнений.

Планируемое время выполнения работы - 8 часов.

## Задание

Создать функцию для вычисления обратной матрицы по методу Гаусса-Жордана. Размер матрицы передавать в функцию в качестве параметра. Для упрощения алгоритма следует присоединить единичную матрицу справа к исходной и выполнять все преобразования над объединенной матрицей размером N\*2N. Обратная матрица получится на месте единичной в столбцах N…2N, а на месте исходной матрицы в столбцах 0…(N-1) должна получиться единичная матрица.

Включить в алгоритм проверку на существование обратной матрицы.

Алгоритм решения системы линейных алгебраических уравнений оформить в виде функции.

Шаги выполнения задания.

Часть 1.

* 1. Разработать схему алгоритма для вычисления обратной матрицы методом Гаусса-Жордана.
  2. Разработать программу, реализующую метод Гаусса-Жордана. Для ускорения процесса разработки программы на этом шаге следует сконцентрироваться на реализации алгоритма и не использовать динамическую память для хранения матриц и не оформлять реализацию алгоритма в виде функции.
  3. Отладить программу, используя контрольный пример.
  4. Оформить алгоритм в виде функции для вычисления обратной матрицы по методу Гаусса-Жордана. Размер матрицы передавать в функцию в качестве параметра.

Часть 2.

* 1. Выполнить тестирование программы. Проверку правильности результатов вычислений выполнять путем умножения полученной обратной матрицы на исходную (в результате должна получиться единичная матрица). Для реализации контроля разработайте функцию, реализующую произведение прямоугольных матриц. В процессе выполнения программы на экран должны выводиться следующие данные: исходная матрица, промежуточные результаты после прямого и обратного ходов, обратная матрица и результат умножения прямой матрицы на обратную. Для печати матрицы использовать функцию из лабораторной работы 9.
  2. Доработать алгоритм для случая, когда на главной диагонали появляется ноль. Для этого в прямом ходе перед делением выполнить проверку на ноль элемента на пересечении столбца коэффициентов исключаемой переменной и главной диагонали исходной матрицы. Если элемент равен 0 (а на него нужно делить элементы текущей строки), то следует поменять местами текущую строку с одной из нижележащих строк, в которой в этом столбце элемент не равен 0. Если таких строк нет, то выдать сообщение: «Обратная матрица не существует».

1.3. Применить функцию вычисления обратной матрицы для решения системы линейных алгебраических уравнений.

1.4. Оценить влияние коэффициентов исходной матрицы на точность вычисления обратной матрицы. В качестве оценки погрешности использовать значения «нулей» единичной матрицы, полученной в результате перемножения исходной матрицы на обратную.

## Рекомендации по выполнению работы

### Алгоритм вычисления обратной матрицы

В математике все доказательства, связанные с получением обратной матрицы, полностью основаны на применении определителей. Этот процесс ясно показывает способ вычисления каждого элемента обратной матрицы, но не эффективен при обращении матриц большого порядка, поскольку требует излишних вычислений.

На практике для получения обратной матрицы используются методы, основанные на решении системы линейных алгебраических уравнений.

В лабораторной работе рассматривается вычисление обратной матрицы методом Гаусса-Жордана. Гаусс предложил метод решения систем линейных алгебраических уравнений путем последовательного исключения переменных, а затем Жордан использовал его для нахождения обратной матрицы, добавив преобразование единичной матрицы.

Прежде чем приступать к программированию, разработаем численный пример получения обратной матрицы методом Гаусса-Жордана. Это позволит лучше понять алгоритм метода и использовать этот пример в качестве контрольного при разработке алгоритма программы.

Предположим, имеется матрица



Метод Гаусса-Жордана предполагает нахождение обратной матрицы путем решения следующей системы уравнений

 (1)

относительно *х, у* и *z* с помощью обычной процедуры последователь­ного исключения (здесь символы *х, у, z* и *а, b, с* характеризуют скалярные величины).

Умножение обеих частей уравнения слева на *А-1* дает решение в виде

, (2)

если A-1 существует.

Следовательно, если решение системы (1) относительно *х, у* и *z* выражено в ***явном*** ***виде*** через a*, b* и *c* и записано в матричной форме, то приводящая к решению матрица равна A-1. Воспроизведем полно­стью этот процесс.

Допустим, что уравнения (1) имеют следующий вид:

*2х + 5у + 7z* = a, (3)

*3х + 9у + 15z = b*, (4)

*5х + 16y + 20z = c.* (5)

Деление (3) на 2 (коэффициент на главной диагонали) дает

*x + 2,5y + 3,5z = 0,5a.* (6)

Исключим *x* из нижележащих строк. Для этого умножим

обе части уравнения (6) на три и вычтем их из (4), а затем,

умножив обе части уравнения (6) на пять, вычтем их

из (5). Тогда

*0+1,5y + 4,5z = b - 1,5a* (7)

и

*0+3,5y + 2,5z = c - 2,5a*. (8)

Разделив (7) на 1,5 и умножив (8) на 2, получим

*у + 3z =* b/1,5 - a*,* (9)

*7y + 5z = 2c - 5a.* (10)

Вычитая (9), умноженное на 7, из (10), мы получим

*-16z = 2c - 5a - 7 (b/1,5 - a).*

Следовательно,

*z = (2a - 7b/1,5 + 2c)/(- 16) = (-3a + 7b - 3c)/24.*  (11)

На этом прямой ход алгоритма Гаусса завершается.

Переходим к обратному ходу.

Подстановка *z* в (9) приводит к следующему результату:

*у = (—15a — 5b + 9c)/24,* (12)

а подстановка *y* и *z* в (6) дает

*x = (60a — 12b — 12c)/24.* (13)

Последние три результата могут быть представлены в матричном виде следующим образом:

=. (14)

Сравнивая этот результат с (2), мы приходим к заключению, что

A-1=.

Умножив обе части этого соотношения на *А,* можно убедиться в спра­ведливости полученного результата. Эта процедура, естествен­но, очень утомительна для человека, но для компьютера она не представляется слож­ной: отдельные шаги расчетов по существу одни и те же как для больших, так и для маленьких мат­риц (для больших их просто больше).

Так как компьютер работает с числами (а не с переменными a*, b*, *c*), то для того чтобы решение системы (1) относительно *x, у* и *z* выразить в ***явном*** ***виде*** через a*, b* и *c* и записать в матричной форме, умножим правую часть системы уравнений (1) на единичную матрицу и будем выполнять над ней те же преобразования, что и над исходной матрицей.

Алгоритм преобразования состоит из двух шагов:

1. Прямой ход. В результате исходная матрица преобразуется в верхнюю треугольную матрицу.
2. Обратный ход. В результате исходная матрица преобразуется в единичную, а введенная единичная матрица – в матрицу, обратную исходной, то есть система уравнений (1) преобразуется в систему уравнений (2).

Выполним с помощью этого алгоритма рассмотренный выше пример:

=.

#### Шаг 1. Прямой ход

Исключаем *x* из нижележащих строк:

 .

Исключаем *y:*

 .

Устанавливаем коэффициент при *z* равным 1:

 .

#### Шаг 2. Обратный ход

Исключаем *z* из всех вышележащих строк:

Исключаем *y* из всех вышележащих строк:

 .

Представим результат в матричном виде:

=.

Сравнив результат с (14), убеждаемся, что все вычисления сделаны правильно.

### Точность вычисления обратной матрицы.

Основной источник погрешностей при выполнении вычислений на ЭВМ – это ошибки округления. Ошибки округления всегда возникают в последнем значащем разряде. Однако накопление ошибок в связи с выполнением большого количества вычислений может повлиять на последние два и более разрядов.

Для уменьшения погрешностей округления нужно использовать более «длинные» типы данных для представления коэффициентов, например, double вместо float.

Влияние погрешностей округления на результат вычислений зависит от используемых методов расчета. Наиболее часто встречающиеся источники увеличения влияния погрешностей округления на результат – деление на малые величины и сложение больших величин с малыми величинами. Полностью предупредить подобные ошибки нельзя, однако в каждом конкретном случае нужно использовать методы расчета, которые сводят к минимуму влияние ошибок. Например, если нужно складывать большое количество малых величин с большой величиной, то нужно сложить сначала все малые величины, а затем их сумму, которая уже не будет такой малой, сложить с большой величиной. Если при сложении последовательно прибавлять малые величины к большой, то при выравнивании порядков малые величины будут превращаться в ноль и не повлияют на результат сложения. Общее правило заключается в том, что надо так организовать процесс вычислений, чтобы, по возможности, все операции выполнялись бы над величинами одного порядка.

Вообще говоря, ЭВМ не делает ошибок, но она выполняет ошибочные программы точно так же, как и правильные. Поэтому оценку точности результатов вычислений должен делать программист. В случае получения обратной матрицы правильность результата вычислений можно оценить путем умножения обратной матрицы на исходную. В результате должна получиться единичная матрица.

## Содержание отчета

## Контрольные вопросы

## Рекомендуемые источники информации

# Лабораторная работа 10 Ввод, сортировка и двоичный поиск в массиве структур.

## Цель и задачи работы

Англо-русский словарь построен в виде массива структур Dictionary и ***хранится в файле***. Структура содержит английское слово и соответствующее ему русское слово.

Размер словаря может изменяться. Для хранения словаря в ОП использовать динамический массив. Размер словаря (количество пар слов) записывается в начало файла.

Разработать программу, которая:

* обеспечивает формирование словаря (добавление и удаление записей);
* считывает словарь из файла;
* записывает словарь, отсортированный по английским значениям слов, в файл;
* обеспечивает просмотр словаря;
* выполняет перевод слов с английского на русский, используя для поиска слова в словаре метод двоичного поиска в отсортированном массиве;
* выполняет перевод слов с русского на английский, используя для поиска слов в словаре метод перебора;

Программа должна обеспечивать диалог с помощью меню.

Начальное число слов в словаре должно быть не меньше 10.

Планируемое время выполнения работы - 6 часов.

## Задание и требования к результатам работы

## Рекомендации по выполнению работы

### Шаги разработки программы

*1. Определение состава и способа представления исходных данных, результатов и промежуточных данных.*

Исходные данные. Так как размер словаря по условию задачи ограничен, и фактическое число записей в словаре может изменяться , то для хранения словаря в оперативной памяти (ОП) будем использовать динамический массив из элементов типа Dictionary.

const int l\_word = 31;

struct Dictionary {

char engl[l\_word]; // слово по-английски

char rus[l\_word]; // слово по-русски

};

Максимальная длина русских и английских слов *l\_word* = 30 символам. Словарь хранится в текстовом файле. Элементы массива структур Dictionary будем записывать в файл последовательно, начиная с нулевого. При этом значения полей структуры записываются в виде отдельных строк, т.е. каждое слово должно заканчиваться символом ‘\0’.

*2. Разработка алгоритма решения задачи.*

При программировании этой задачи уделим дополнительное внимание разбиению на функции и спецификации их интерфейсов. Например, логично оформить в виде функции каждую операцию со словарем (формирование, поиск, добавление и уда­ление элемента), поскольку они представляют собой законченные действия.

Интерфейс пользователя организуем в виде простейшего меню, которое будет выводиться на экран после каждого действия. Меню представим в виде пронумерованного перечня возможных действий пользователя, красиво размещенного на экране, а выбор действия будем выполнять путем ввода его номера в перечне. В функции *menu()* следует предусмотреть реакцию на ввод пользователем непредусмотренных алгоритмом данных, например, буквы вместо цифры (так называемая «защита от дурака»).

Будем исходить из того, что все функции должны быть независимы, чтобы изме­нения в одной функции не могли влиять на поведение другой. Для этого всё, что функциям необходимо получать извне, будем передавать им через параметры (а не через глобальные переменные). Прежде всего определим интерфейс нашей программы. В соответствии с заданием, кажется логичным предоставить пользователю следующие возможности:

1. добавление слов в словарь;
2. удаление слов из словаря;
3. перевод слов с английского на русский;
4. перевод слов с русского на английский;
5. просмотр словаря (вывод на экран словаря из ОП);
6. вывод словаря в файл;
7. чтение словаря из файла;
8. выход.

Каждый пункт этого меню, кроме последнего, оформим в виде отдельной функ­ции. Определим прототипы предложенных функций.

*Меню*. Эта функция (в качестве подсказки) выводит пронумерованный перечень возможных действий пользователя и выполняет ввод номера выбранного действия. Это число она должна вернуть в вызвавшую функцию.

int menu( );

*Добавление* *слов в словарь.* Чтобы добавить элемент Dictionary в словарь, надо задать пару слов (элемент массива) и вставить элемент так, чтобы массив оставался отсортированным. Ввод и вставку добавляемого элемента выполним внутри функции, а в функцию передадим указатель на начало словаряи его размерность. Назовем ее addWord.

void addWord(Dictionary \* dict, int & n);

Если слова для добавления могут не только вводиться с клавиатуры, но браться из других источников, и размер словаря при вводе может изменяться, то прототип функции может быть таким:

void addWord(Dictionary \* dict, int & n, Dictionary elem);

По аналогии определите прототипы остальных функций самостоятельно.

*3. Кодирование и тестовые примеры.*

В этой работе удобно использовать технологию создания программы «сверху вниз»: сначала отладить главную функцию, а затем постепенно добавлять к ней остальные. На месте еще не добавленных функций обычно ставятся так на­зываемые *заглушки* — функции, единственный действием которых является вы­вод сообщения о том, что эта функция была вызвана. Первой добавляемой функцией должна быть функция Menu.

При использовании меню главная функция сильно упрощается: она периодически вызывает меню, анализирует запрошенный код операции и вызывает функцию, которая выполняет эту операцию. При выборе операции с номером 7 выполнение программы завершается.

int main(){

Dictionary\* dict; //массив структур для хранения словаря в оперативной памяти

int num\_w=0; // число записей в словаре

while (true) {

switch (menu()) {

case 1:addWord(dict, num\_w); break;

case 2:

case 3:

case 4: break;

case 5: printDict(dict,num\_w); break;

case 6: break;

case 7: return 0;

default: cout<<" Надо вводить число от 1 до 7"<<endl; break;

}

}

return 0;

}

После отладки menu(), запрограммируйте и отладьте функции addWord(…) и printDict(…).

addWord(…) добавляет записи в массив структур, а printDict(…) распечатывает массив из оперативной памяти.

### Работа со структурами

Структуры в C++ обладают практически теми же возможностями, что и классы, но чаще их применяют просто для логического объединения связанных между со­бой данных. В структуру, в противоположность массиву, можно объединять дан­ные различных типов.

Например, требуется обрабатывать информацию о расписании работы конференц-зала, и для каждого мероприятия надо знать время, тему, фамилию организатора и количество участников. Поскольку вся эта информация относится к одному собы­тию, логично дать ему имя, чтобы впоследствии можно было к нему обращаться. Для этого описывается новый тип данных (обратите внимание на то, что после описания стоит точка с запятой):

struct Event {

int hour, min;

char theme[100], name[100];

int num;

};

Имя этого типа данных — Event. Можно описать переменные этого типа точно так же, как переменные встроенных типов, например:

Event el, e2[10]; // структура и массив структур

Переменные структурного типа можно размещать и в динамической области па­мяти, для этого надо описать указатель на структуру и выделить под нее место:

Event \*pe = new Event; // структура

Event \*pm = new Event[m]: // массив структур

Элементы структуры называются *полями.* Поля могут быть любого основного типа, массивом, указателем, объединением или структурой. Для обращения к полю ис­пользуется *операция выбора* («точка» (.) для переменной и -> для указателя), на­пример:

Структуры одного типа можно присваивать друг другу:

\*ре = el; pm[l] = el; pm[4] = e2[0];

Но присваивание - это и все, что можно делать со структурами целиком. Другие операции, например сравнение на равенство или вывод, не определены. Впрочем, пользователь может задать их самостоятельно, поскольку структура является ви­дом класса, а в классах можно определять собственные операции. Мы рассмотрим эту тему во втором семестре.

*Ввод/вывод структур,* как и массивов, выполняется поэлементно. Вот, например, как выглядит ввод и вывод описанной выше структуры e1:

cin >> e1.hour >> e1.min;

cin. getline (e1. theme, 100);

cout << e1.hour << ' ' << e1.min << ' ' << e1.theme << endl;

Структуры (но не динамические) можно инициализировать перечисле­нием значений их элементов:

Event е3 = {13, 50, "Работа со структурами ", "Петрова",25};

### Дополнительные требования для «сильных» студентов:

* использовать для хранения словаря бинарный файл;
* сделать два режима работы со словарем: англо-русский и русско-английский,

в каждом из которых используется двоичный поиск. Для выполнения этого требования разработать «универсальную» функцию сортировки, которая могла бы сортировать слова в англо-русском и русско-английском словарях, используя для сравнения слов в алгоритме сортировки функцию-предикат типа

bool Compare(char\* word1,char\* word2).

Функция Compare() должна передаваться в функцию сортировки в качестве параметра.

## Содержание отчета

## Контрольные вопросы

## Рекомендуемые источники информации

Вирт Н. Алгоритмы+структуры данных=программы. М.:Мир, 1985. (В этой монографии подробно рассматриваются алгоритмы сортировки, рекурсивные алгоритмы и динамические типы данных. Изложение базируется на языке Паскаль, но излагаемый материал во многом применим и к процедурному программированию на Си++.)

Страуструп Б. Язык программирования С++. Вторая редакция. К.:"ДиаСофт", 1993. ("Классическое" справочное руководство по языку Си++, написанное автором языка. Эта книга пригодится тем, кто собирается в будущем серьезно заниматься программированием на Си++.)

# Домашнее задание. Методические указания к домашнему заданию по курсу «Основы программирования»

## Цели домашнего задания

Целью домашнего задания является закрепление приобретенных в процессе выполнения лабораторных работ практических навыков реализации на языке С++ следующих приемов структурного программирования:

1. Использование методов структурного проектирования программ при разработке алгоритма выполнения задания.
2. Использование структур и массивов из структур для хранения данных.
3. Использования методов структурного программирования при разработке библиотеки базовых функций для реализации отдельных блоков алгоритма.
4. Организация обмена данными между функциями с использованием указателей и ссылок.
5. Создание и обработка динамических массивов.
6. Отладка и тестирование программ.

## Требования к выполнению задания

Ввести с клавиатуры текст, состоящий из нескольких предложений. Текст состоит из латинских (или русских) букв (строчных и прописных), пробелов и знаков пунктуации (точки, запятые и др.) и заканчивается символом "конец текста" (смотри таблицу кодов ASCII). Предложения в тексте разделяются пробелами. Количество предложений и их содержание должно обеспечивать проверку правильности выполнения задания, то есть являться тестом для разработанной программы.

БЕЗ ИСПОЛЬЗОВАНИЯ типа STRING:

* разработать функцию для ввода такого текста с клавиатуры. Функция должна контролировать ввод, чтобы вводить только указанные в задании символы и возвращать строку с введенным текстом;
* разработать структуры Text и Sentence для хранения введенного текста, которые обеспечивали бы возможность обращения к символам текста как к элементам трехмерного массива, индексами которого являются номер предложения в тексте, номер слова в предложении и номер символа в слове;
* разработать функцию, которая преобразует введенный текст в массив из предложений;
* разработать функцию, которая преобразует предложение в массив из слов;
* поместить определения этих функций и структур, которые являются базовыми для ДЗ (так как могут использоваться для выполнения любого варианта задания), в отдельный модуль (файл) и подключить его к программе;
* заполнить структуру Text введенным текстом;
* выполнить в соответствии с индивидуальным заданием преобразование данных в структуре Text;
* распечатать преобразованный текст. Печать предложения начинать с новой строки, слова печатать через один пробел, сохраняя в предложении использованные знаки пунктуации.

*Задание выполняется группой из 3-х человек, один из которых выбирается руководителем (смотри п.2.1* *методических указаний).*

Прежде чем приступать к разработке базовых функций для выполнения ДЗ, оцените свои возможности реализации выбранного вами решения с целью правильно рассчитать своё время. Если затраты на разработку кажутся слишком большими, следует выбрать более простой, но и более дешевый (в смысле максимальной оценки) способ выполнения задания. Правила оценки ДЗ приведены в п.4 методических указаний.

### Групповая разработка проектов

Сложные проекты программирования обычно выполняются на различных стадиях разными людьми. Только разработку небольших проектов ведет один специалист, выполняющий самостоятельно разработку алгоритмов, кодирование и отладку всех частей программы. Поэтому необходимо учиться умению работать вместе с другими людьми и находить с ними общий язык. Это обязательное условие успешного выполнения проекта.

 Групповая разработка домашнего задания может стать хорошим средством комплексной проверки эффективности методов программирования, которые вы использовали в лабораторных работах. Кроме того, групповая разработка создает возможность для более творческого подхода студентов к процессу программирования при выполнении индивидуальных заданий.

Каждая группа должна разработать один проект, который позволял бы выполнять индивидуальные задания каждого из членов группы с использованием библиотеки базовых функций. Каждая группа создает свою библиотеку, содержащую определения структур Text и Sentence и базовые функции, выполняющие ввод текста и заполнение структур Sentence и Text. Базовые функции, разрабатываемые членами группы, должны быть одинакового типа, согласованного с руководителем группы, но иметь свои имена и реализацию, то есть иметь одинаковую сигнатуру. Так как имя функции включается в прототип, то для того, чтобы программа могла работать с различными комбинациями функций, вызовы функций в программе следует выполнять через указатель на функцию. Например, для первой группы

char\* InputText1();

Text& GetText1(char\*);

Sentence& GetSentence1(char\*);

Затем функции членов группы объединяются в общую библиотеку, которая хранится в отдельном модуле.

Сначала совместно разрабатывается скелет программы и прототипы базовых функций. Каждый член группы должен сам разработать часть функций и знать реализацию ***всех базовых функций,*** разработанных другими членами группы. Всякое изменение в проекте, касающееся других членов группы, например, изменение типов функций, должно быть согласовано с руководителем и другими членами группы.

### Шаги выполнения задания

1. Совместно определить состав функций и данных, включая структуры, которые бы могли быть использованы каждым членом группы для выполнения индивидуального задания. Разработать прототипы функций.
2. Разработать укрупненную схему алгоритма, определяющего использование функций для выполнения индивидуальных заданий. На схеме алгоритма в блоках вызова функций должны быть записаны их заголовки с указанием идентификаторов параметров, чтобы можно было бы проследить связь функций по параметрам.
3. Разработать функцию ***main(),*** которая позволяла бы протестировать выполнение индивидуального задания ***как с базовыми функциями своей группы, так и с базовыми функциями других групп***.

Пример главного модуля, использующего структуру Text

#include<conio.h>

#include<iostream>

#include<iomanip>

#include"dzLibrary1.cpp"//модуль с базовыми функциями всех //членов группы, обеспечивающими загрузку вводимого текста в //структуруText:

//char\* InputText();

//Text& GetText(char\*);

//Sentence& GetSentence(char\*);

usingnamespacestd**;**

//----------------------------------------------------

int menu( );

//Пример структур **Text и Sentence** (с динамическими массивами):

struct **Sentence{**

char\*\* words;//массив из указателей на слова в предложении

int **size;** //размер массива (число слов в предложении)

**};**

struct **Text {**

**Sentence \*sentences;**//массив из предложений в тексте

int **size;** //размер массива (число предложений)

**};**

Text& loadText();

void dz1(Text &text);

void dz2(Text &text);

<прототипы функций, реализующих варианты других членов группы>

//===============================

int main()

{

system("chcp 1251 >nul");

char\* str=" ";

Text text;

while (true)

{

switch (menu())

{

case 0: text = loadText(); break;

case 1: dz1(text); break;

case 2: dz2(text);break;

<вызовы функций, реализующих варианты других членов группы>

case30**:** return0**;**

default: cout<<("Надо вводить числа от 1 до 30\n");

break;

}

}

system("pause");

return 0;

}  
<реализация функций menu(), loadText(),dz1(**),…** dz23()…>

1. Выполнить кодирование и отладку базовых функций и функций для выполнения индивидуального задания. В процессе выполнения этого шага может возникнуть необходимость внесения изменений в результаты 1-го шага. Все изменения должны быть согласованы с руководителем и остальными членами группы. При разработке программы группа должна придерживаться единого стиля программирования (именование переменных, отступы, комментарии).
2. Выполнить тестирование разработанной программы.

6. Для тестирования правильности выполнения задания преподавателем должны быть разработаны соответствующие контрольные тексты.

## Требования к отчету

Отчет по лабораторной работе должен состоять из 4-х разделов, отражающих основные этапы разработки программы:

- Постановка задачи;

- Разработка алгоритма;

* Кодирование и отладка (соответствующий раздел отчета называется «Текст программы»);

- Тестирование.

В разделе «Постановка задачи» должен быть приведен текст задания, согласованные с преподавателем (заказчиком) уточнения, если они требуются. Раздел завершается описанием структур данных и прототипов функций, которые предлагается использовать в ДЗ.

Раздел «Разработка алгоритма» должен содержать следующую информацию:

* краткое описание (обоснование) алгоритма;
* описание входных, выходных и вспомогательных данных с указанием их идентификаторов и типов;
* схему алгоритма, состоящую из двух частей: общей (укрупненной) схемы и уточненной схемы блока, реализующего преобразование слов в предложении в соответствии с вариантом задания.

В разделе «Текст программы» должен быть приведен листинг программы, включающий необходимые комментарии.

В разделе «Тестирование» должна быть приведены распечатки экрана с тестовыми предложениями, достаточными для тестирования программы.

Отчет должен быть распечатан на принтере на листах бумаги формата А4, скрепленных в левом верхнем углу с помощью степлера, и подписан исполнителем с указанием даты сдачи отчета преподавателю.

## Оценка выполнения задания

Максимальная оценка –10 баллов.

Минимальная оценка – 4 балла, ставится при использовании в ДЗ функции ввода текста (основная базовая функция) и функции выполнения индивидуального задания.

Добавление баллов производится за использование для ввода текста следующих дополнительных базовых функций и структур:

1 - функций для загрузки вводимого текста в структуры Sentence и Text, которые

должны обеспечивать обращение к символам в тексте как к элементам трехмерного массива. В структурах Sentence и Text используются статические массивы +3 балла;

2 - плюс перегрузка операции вывода в поток (<<) для структур Sentence и Text +1 балл;

3 - плюс перегрузка операции индексации ([]) для структур Sentence и Text +1 балл;

4 - плюс использование динамических массивов в структурах Sentence и Text +1 балл.

Например, если вы выполнили два первых пункта дополнительных условий, ваша оценка будет 8 баллов, а если все четыре – то 10 баллов.

## Варианты заданий

### Вариант 1

Оставить в предложениях слова, отличающиеся от последнего слова предложения, перенеся в них первую букву в конец слова.

### Вариант 2

Оставить в предложениях слова, отличающиеся от последнего слова предложения, удалив из них последнюю букву.

### Вариант 3

Оставить в предложениях слова, отличающиеся от последнего слова предложения, удалив из них все последующие вхождения первой буквы.

### Вариант 4

Оставить в предложениях слова, отличающиеся от последнего слова предложения, удалив из них все предыдущие вхождения последней буквы.

### Вариант 5

Оставить в предложениях слова, отличающиеся от последнего слова предложения, оставив в них только первые вхождения каждой буквы.

### Вариант 6

Оставить в предложениях слова, отличающиеся от последнего слова предложения, и если слово нечетной длины, удалить его среднюю букву.

### Вариант 7

Оставить в предложениях слова, отличающиеся от последнего слова предложения, и одновременно симметричные.

### Вариант 8

Оставить в предложениях слова, отличающиеся от последнего слова предложения, в которых первая буква слова входит в него еще только один раз.

### Вариант 9

Оставить в предложениях слова, отличающиеся от последнего слова предложения, в которых буквы упорядочены по алфавиту.

### Вариант 10

Оставить в предложениях слова, отличающиеся от последнего слова предложения и совпадающие с начальным отрезком алфавита (a, ab, abc и т.д.).

### Вариант 11

Оставить в предложениях слова, отличающиеся от последнего слова предложения и совпадающие с конечным отрезком алфавита (z, yz, xyz и т.д.).

### Вариант 12

Оставить в предложениях слова, отличающиеся от последнего слова предложения и имеющие максимальную длину.

### Вариант 13

Оставить в предложениях слова, отличающиеся от последнего слова предложения и не содержащие повторяющихся букв.

### Вариант 14

Оставить в предложениях слова, отличающиеся от последнего слова предложения, в которые каждая буква входит не менее двух раз.

### Вариант 15

Оставить в предложениях слова, отличающиеся от последнего слова предложения, в которых гласные буквы (a, e, i, o, u, y) чередуются с согласными.

### Вариант 16

Оставить в предложениях слова, отличающиеся от последнего слова предложения, изменив в них порядок букв на обратный.

### Вариант 17

Оставить в предложениях слова, отличающиеся от последнего слова предложения, расположив их в алфавитном порядке.

### Вариант 18

Оставить в предложениях слова, отличающихся от последнего слова предложения и если слово оканчивается на **ov, ova, in, ina, ev,** **eva,** заменить это окончание на **idze**.

### Вариант 19

Оставить в предложениях слова, отличающиеся от последнего слова предложения, поменяв местами с предыдущим словом слова, стоящие на чётном месте.

### Вариант 20

Оставить в предложениях слова, отличающиеся от последнего слова предложения, заменив на **xxx** каждое слово из 5 букв, имеющее в середине букву **z**.

### Вариант 21

Оставить в предложениях слова, отличающиеся от последнего слова предложения, заменив на букву **u** гласные, начиная со второй.

### Вариант 22

Оставить в предложениях слова, отличающихся от последнего слова предложения, заменив в них каждую букву, стоящую в алфавите на месте k, на букву, стоящую в алфавите на месте k с конца алфавита (при замене сохранить регистр).

### Вариант 23

Оставить в предложениях повторяющиеся слова, удалив все повторные вхождения слова. Если таких слов нет, то удалить предложение из текста.

### Вариант 24

Оставить в предложениях слова, входящие в предложение по одному разу. Если таких слов нет, то удалить предложение из текста.

### Вариант 25

Оставить в предложениях все различные слова текста, указав для каждого из них количество его вхождений в текст.

## Вспомогательные материалы для выполнения лабораторных работ

1. Цезарь для шифрования своих посланий использовал следующий прием. При кодировании и декодировании писем он заменял буквы в письме на следующие по алфавиту буквы с постоянным смещением, равным 13, то есть к порядковому номеру буквы в латинском алфавите, содержащим 26 букв, он прибавлял 13 и получал порядковый номер буквы в кодируемом (декодируемом) письме. Если полученный таким образом номер буквы был больше 26, то он уменьшался на 26. [↑](#footnote-ref-1)
2. Если во вводимой строке больше символов, чем может вместить выделенная для ее хранения об­ласть, поведение программы не определено. Скорее всего, она завершится аварийно [↑](#footnote-ref-2)
3. Символ перевода строки '\n' появляется во входном потоке, когда вы нажимаете клавишу Enter [↑](#footnote-ref-3)