МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

А.В. ГАЙДЕЛЬ, А.Г. ХРАМОВ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО КУРСУ «ОСНОВЫ ИНФОРМАТИКИ»

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве практикума для обучающихся по основной образовательной программе высшего образования по направлению подготовки 01.04.02 Прикладная математика и информатика

САМАРА
Издательство Самарского университета
2019

УДК 004(075)
ББК 32.97я7
Г14

Рецензенты: д-р техн. наук, проф. С. В. С м и р н о в,
д-р техн. наук, проф. В. А. Ф у р с о в

Гайдель, Андрей Викторович

Г14 **Лабораторный практикум по курсу «Основы информатики»:** практикум / *А.В. Гайдель, А.Г. Храмов.* – Самара: Изд-во Самарского университета, 2019. – 172 с.

ISBN 978-5-7883-1416-7

Содержит учебно-методические материалы для выполнения лабораторных работ по курсу «Основы информатики». Всего представлено 4 лабораторных работы, затрагивающих основные навыки программирования. Для каждой лабораторной работы приведены необходимые теоретические сведения, примеры реализации программ на языке C++, а также по 30 вариантов задания, кардинально отличающихся друг от друга решаемой задачей.

Предназначен для подготовки бакалавров по направлению подготовки 01.03.02 Прикладная математика и информатика в рамках образовательной программы «Компьютерные науки».

Подготовлен на кафедре технической кибернетики.

УДК 004(075)
ББК 32.97я7

ISBN 978-5-7883-1416-7

Оглавление

Введение

Информатика – это наука об информации, изучающая способы её сбора, хранения, обработки и передачи. Предметом исследования информатики являются всевозможные вычислительные устройства и сети, а также процессы, возникающие при их участии. Среди прочих в информатике решаются задачи об оптимальном хранении данных, об организации доступа к ним, о наиболее эффективной реализации вычислительных процессов для обработки данных определённым образом, об удобном взаимодействии между человеком и вычислительными устройствами и многие другие.

Отечественный термин «информатика» в современном понимании соответствует англоязычному термину «*Computer science*»: это наиболее общая наука о вычислительных устройствах и обо всём, что с ними связано. Под английским словом «*Informatics*» понимают в основном дисциплины, смежные непосредственно с теорией информации. Это создаёт некоторую путаницу при переводах текстов в обе стороны.

Все эти термины возникли в середине XX века. Изначально они обозначали более узкие дисциплины, но с развитием науки получили довольно высокое значение. Со времени своего возникновения, год от года информатика становилась всё более и более важной и актуальной наукой. Компьютерная техника развивалась, вычислительные устройства всё больше внедрялись в обиход. В итоге сегодня информатика – одна из наиболее значимых научных дисциплин. Именно благодаря результатам, полученным в этой науке, современные люди способны повсеместно получать доступ к любой информации и обмениваться найденной информацией друг с другом.

Информатика – это в большинстве своём прикладная наука. Она изучает процессы и явления, непосредственно протекающие в реаль-

ности, и решает задачи, связанные с ними. В частности, в информатике, кроме прочего, изучаются базы данных, языки программирования, архитектура компьютера, машинное зрение и искусственный интеллект.

Именно поэтому изучение информатики невозможно без постоянной практики, ведь это прикладная теория и именно практические результаты делают её настолько распространённой и значимой. Вот почему важно самостоятельно создавать программы для вычислительных устройств и анализировать их работу в рамках лабораторных занятий. В данном издании предлагается несколько заданий для лабораторных работ, включающих манипуляции с основными типами и структурами данных. При написании этого издания ставилась цель заинтересовать читателя основными задачами, возникающими в информатике, и мотивировать его к дальнейшему самостоятельному решению подобного рода задач.

Текст издания разбит на главы, каждая из которых соответствует одной лабораторной работе. Перед каждой работой приведены некоторые сопутствующие теоретические сведения, а также примеры реализации. В каждой главе приведено по 30 качественно различающихся заданий для соответствующей лабораторной работы. Каждое задание заключается в написании компьютерной программы, считывающей некоторые данные, выполняющей с ними определённые действия, и выводящей результат работы на экран. Нет требования, определяющего конкретный язык программирования, на котором необходимо писать эти программы, но для определённости большая часть примеров приведена на языке C++. В конце каждой главы приведены контрольные вопросы, знание ответов на которые позволит успешно выполнить лабораторную работу.

Требования к лабораторным работам

Лабораторная работа выполняется группой от одного до четырёх человек. Все студенты в одной команде несут полную ответственность за лабораторную работу и получают за неё одинаковые оценки. Вариант задания назначается случайным образом. Результатом выполнения лабораторной работы является исходный код программы на одном из языков программирования. Успешно сданная лабораторная работа оценивается вещественным числом от 0 до 1. Успешная сдача всех лабораторных работ является необходимым условием для допуска к экзамену.

Программа должна обладать интерфейсом пользователя и осуществлять корректное взаимодействие с пользователем. Это не обязательно должен быть оконный или псевдографический интерфейс: достаточно консольного интерфейса или, в отдельных случаях, даже использования в качестве интерфейса параметров командной строки. Программа должна корректно завершать работу в случае ввода пользователем корректных входных данных.

Сдавать лабораторную работу можно только на лабораторных занятиях по расписанию. За одно занятие нельзя дважды сдавать одну и ту же лабораторную работу. У каждой лабораторной работы имеется срок сдачи. В случае сдачи лабораторных работ в срок студент может быть освобождён от решения дополнительных задач на экзамене.

В процессе сдачи лабораторной работы преподавателю предъявляется исходный код и запущенная программа для тестирования её работы. Преподаватель может тестировать программу на любых входных данных, а также задавать выполнившим работу студентам вопросы по исходному коду и по теории, связанной с выполненной работой. Работа считается успешно сданной, если:

– программа успешно работает на любых адекватных входных данных;

– студенты, выполнявшие работу, правильно ответили на все вопросы преподавателя.

Таким образом, лабораторная работа не засчитывается, если хотя бы на одном наборе корректных входных данных программа выдала неверный результат или некорректно завершила работу, а также, если хотя бы на один вопрос, указанный преподавателем, студент, входивший в группу выполнявших лабораторную работу, не смог дать правильный ответ.

Преподаватель оценивает по трёхбалльной системе («плохо», «нормально», «хорошо») следующие параметры выполненной работы.

– Работоспособность (Р) – правильность работы программы на корректных входных данных. В случае плохой оценки этого параметра работа считается несданной.

– Надёжность (Н) – способность программы правильно реагировать на некорректные данные и ошибочные действия пользователя.

– Эффективность (Э) – разумность использования ресурсов компьютера (например, процессорного времени и памяти).

– Пользовательский интерфейс (И) – удобство взаимодействия пользователя с программой.

– Стиль кода (С) – читабельность и эстетическая привлекательность исходного кода программы.

– Ответы на вопросы (О) – правильность ответов на вопросы преподавателя. В случае плохой оценки этого параметра работа считается несданной.

Оценка за лабораторную работу складывается как средняя оценка по всем вышеуказанным параметрам. В случае выполнения работы группой из трёх студентов, каждый из них получает 2/3 оценки, а в случае выполнения работы группой из четырёх студентов каждый из них получает только половину итоговой оценки. Рекомендуется выполнять работу группой из одного или двух студентов.

В заданиях намеренно не указываются ограничения на входные данные, их формат, ограничения на использование времени и памяти. При выполнении задания можно самостоятельно выбрать для се-

бя эти ограничения. Это позволяет успешно сдавать как эффективные и универсальные программы, так и неэффективные или работающие только для ограниченного набора входных данных. Однако если программа не работает в тривиальных случаях или для достаточно широкого набора входных данных, она считается неработоспособной и лабораторная работа не засчитывается. Окончательное решение остаётся за преподавателем.

При выполнении лабораторной работы не запрещается пользоваться внешними источниками данных, чужим исходным кодом, использованным на законных основаниях, готовыми библиотеками и любой функциональностью, встроенной в стандартную библиотеку языка программирования. Однако при сдаче лабораторной работы студент несёт полную ответственность за весь исходный код своей программы и за исходный код всех нестандартных библиотек, так что лабораторная работа не будет зачтена в случае непонимания принципов работы любого участка исходного кода, в том числе взятого из примеров исходного кода, приведённых в данном издании. Также нельзя использовать стандартные функции, которые непосредственно делают то, что указано в задании на лабораторную работу. В этом случае требуется реализовать решение самостоятельно.

1 Целочисленная арифметика

1.1 Язык C++

C++ – это универсальный мультипарадигменный компилируемый язык программирования высокого уровня со статической типизацией, предназначенный для решения широкого класса задач. Существуют несколько реализаций компиляторов для этого языка программирования и ещё больше сред разработки, подходящих для большинства известных операционных систем и вычислительных устройств. Широкие возможности языка C++ отчасти обусловлены его богатой стандартной библиотекой, в которой имеются реализации многих сильных структур данных и алгоритмов.

Типичная программа на C++ представляет собой объявление и определение различных структур данных и функций для работы с ними. Функции описываются императивно в виде последовательности инструкций, исполняемых друг за другом. Исходный код программы как обычно пишется в текстовых файлах, которые подвергаются компиляции и последующей компоновке в единый исполняемый файл. На рис. 1 представлена схема сборки программы на C++ от текстовых файлов с исходным кодом до исполняемого файла.

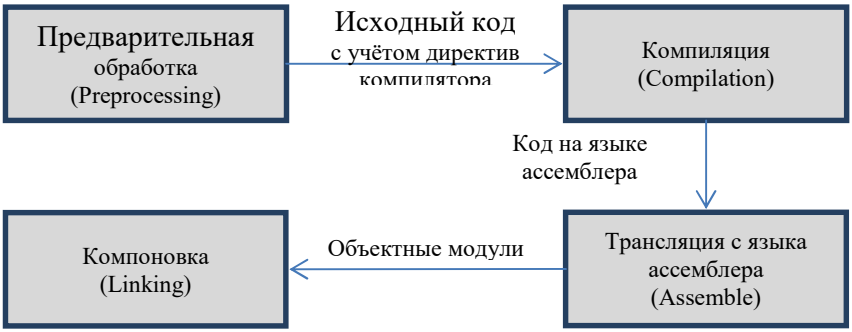

Рис. 1. Сборка программы на C++

Сначала с помощью препроцессора обрабатываются директивы компилятора. После этого полученный исходный код компилируется в исходный код низкого уровня на подобии языка ассемблера. Язык ассемблера легко транслируется в машинный код, и получаются объектные модули, содержащие специальным образом аннотированный машинный код, понятный вычислительному устройству. Разные объектные модули связываются между собой компоновщиком, и в итоге получается исполняемый файл, который можно запустить в некоторой операционной системе.

В листинге 1 приведён исходный код программы «Hello, World!», которая в результате запуска выводит в консоль сообщение «Hello, World!». Это стандартный пример первой программы, полезный при изучении любого языка программирования. Как видно, даже простейшая программа на C++ выглядит весьма непросто, если заранее не знать ничего об этом языке.

Листинг 1. Программа «Hello, World!» на C++

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

В первой строке содержится *директива компилятора* «`#include`», сигнализирующая, что вместо неё нужно вставить исходный код, расположенный в файле «`iostream`». Это файл *стандартной библиотеки* C++, содержащий инструменты для консольного ввода и вывода данных. Вторая строка оставлена пустой для красоты в соответствии со стилем написания исходного кода.

В третьей строке объявляется *функция* «`main`». Это основная функция, являющаяся *точкой входа* в программу. При запуске программы исполняется именно эта функция. Пустые скобки после имени этой функции означают, что она не принимает никаких аргумен-

тов, а ключевое слово «int» перед ним означает, что функция возвращает целое число.

Далее в фигурных скобках записано *тело* функции main. Первая строка этого тела содержит, собственно, вывод на экран сообщения. Для этого используется *поток вывода* «cout», расположенный в *пространстве имён* «std», которому с помощью операции вывода «<<» передаётся сначала строковый *литерал* «"Hello, World!"», а затем таким же образом признак конца строки «endl», также расположенный в пространстве имён «std». Идентификаторы «cout» и «endl» как раз и объявлены в файле «iostream». Каждая отдельная инструкция завершается символом «;».

В последней строке тела функции main содержится *оператор* return, немедленно возвращающий из функции целочисленный литерал 0. Это и есть целое число, которое возвращает функция main. Это число обозначает *код возврата* программы, который интерпретируется операционной системой после завершения работы этой программы. Нулевой код возврата означает, что программа выполнилась и завершилась успешно. Любой другой код возврата означает, что в ходе выполнения программы произошла ошибка, и программа была завершена принудительно. *Никогда* намеренно не следует возвращать ненулевой код возврата, по крайней мере, при выполнении лабораторных работ.

Другой пример простой программы, популярный при первом изучении языков программирования, – это программа «A+B». Исходный код подобной программы на языке C++ приведён в листинге 2. Эта программа решает задачу сложения двух целых чисел, введённых пользователем. После запуска она ожидает ввода в консоль двух целых чисел через пробел или перевод строки, складывает их и выводит в консоль их сумму.

Листинг 2. Программа «A+B» на C++

```
#include <iostream>


int main()
{
    int a, b;
    std::cin >> a >> b;
    std::cout << a + b << std::endl;
    return 0;
}
```

В этой программе объявляются две *переменные* `a` и `b`, которые будут хранить два целых числа, введённых пользователем. После их объявления происходит считывание двух целых чисел с клавиатуры в эти две переменные. Для этого используется поток ввода «`cin`», расположенный в пространстве имён «`std`» и объявленный в заголовочном файле «`iostream`». По аналогии с выводом на экран, считывание происходит с помощью операции «>>».

На экран выводится значение *выражения* «`a + b`». Как видно, выражения в языке C++ записываются в обычной инфиксной нотации, в них могут принимать участие переменные и арифметические операции. Выражение возвращает своё значение, которое используется для вывода в поток вывода `cout`. После любого вывода в консоль логично выводить перевод строки, чтобы следующий вывод находился на другой строке.

Теперь становится ясно, почему язык C++ считается сложным для освоения новичками в программировании. Даже чтобы написать простейшую программу на этом языке, требуется знание и понимание многих концепций, которые при первом знакомстве с программированием следовало бы изучать постепенно. Именно поэтому незнакомым с программированием людям придётся приложить существенные усилия, если первым языком программирования, который они стали изучать, стал язык C++.

C++, как и большинство других языков программирования, чувствителен к регистру. Это означает, например, что идентификаторы «`main`» и «`Main`» в нём считаются разными идентификаторами. Пробелы, символы табуляции и переводы строк в этом языке игнорируются компилятором и служат лишь для отделения лексем друг от друга. Это значит, что несколько операторов можно писать в одну строку, что, однако, не рекомендуется делать из-за нарушения стиля написания исходного кода.

Сильная статическая типизация означает, что каждая переменная в этом языке имеет строго определённый тип данных, указанный при её объявлении. Переменная не может изменить свой тип данных, а также в любой точке программы можно легко выяснить, к какому типу данных относится каждая переменная. Такой подход к типизации позволяет обнаруживать множество ошибок при написании программ ещё на этапе компиляции, что облегчает отладку уже написанных программ.

К достоинствам языка C++ можно отнести следующие его особенности.

– Популярность. По сей день больше всего исходного кода программ написано именно на языках C и C++.

– Универсальность. C++ подходит для решения крайне широкого класса задач в программировании, тогда как другие языки занимают определённые ниши и созданы для решения конкретных задач.

– Вычислительная производительность. За счёт возможности контроля ресурсов на достаточно низком уровне и отсутствия многих гарантий программы на C++ опережают по времени исполнения аналогичные программы на других языках программирования.

– Гибкость. Наличие макросов и языка шаблонов позволяет создавать весьма оригинальные стили исходного кода, меняя сам язык до неузнаваемости. Грамотное использование этих инструментов позволяет настроить язык под конкретный проект, существенно повышая удобство написания исходного кода. Кроме того, это позволя-

ет осуществлять тотальный контроль за реализацией узких мест в программе на низком уровне, но при этом не вдаваться в детали реализации не столь значимых участков кода.

Несмотря на наличие очевидных достоинств, язык C++ не лишён следующих недостатков.

– Сложность освоения. Как уже было отмечено ранее, изучение языка C++ сопряжено с существенными трудностями, особенно для людей не знакомых с программированием.

– Сложность написания программ. Универсальность и гибкость языка, а также многочисленные случаи неопределенного поведения приводят к определённой вседозволенности при написании исходного кода, из-за чего особенно легко совершать трудно обнаруживаемые критические ошибки, представляющие серьёзную опасность при исполнении программы.

– Ручное управление памятью. В отличие от большинства языков программирования, в C++ отсутствует *сборщик мусора*, в результате чего программист вынужден самостоятельно следить за освобождением всей выделенной вручную памяти. Это позволяет лучше контролировать работу с памятью на низком уровне, но влечёт дополнительные ошибки, такие как *утечки памяти*.

– Сильная контекстная зависимость. Наличие макросов и возможностей отложенного определения программных объектов влечёт множество неочевидных зависимостей при написании исходного кода, которые тяжело отследить даже при наличии специализированных сред разработки. Использование одних и тех же синтаксических конструкций в разных местах нередко ведёт к различным результатам.

– Медленная сборка программы. Сложность и сильная контекстная зависимость языка приводят к сложности в реализации эффективных компиляторов и компоновщиков. Таким образом, достаточно объёмные промышленные программы на C++ могут собираться часами.

– Низкая читабельность исходного кода. Исходный код на C++ зачастую сложно не только писать, но и читать. Высокая гибкость языка и отсутствие единого соглашения о стиле написания исходного кода приводят к наличию множества стилей и особенностей в написании исходного кода, делающих отдельные участки кода крайне сложными для восприятия посторонним человеком, даже хорошо знакомым с синтаксисом языка C++.

Сам по себе язык C++ определяется стандартами ISO, последний их которых получил неофициальное название C++17 и на момент написания этого документа находится в стадии разработки, но в скором времени должен быть опубликован. Наиболее популярные реализации компилятора для языка C++ – Microsoft Visual C++ Compiler, Clang и GNU C++ Compiler, входящий в состав GNU Compiler Collection. Все эти компиляторы уже поддерживают многие нововведения стандарта C++17, однако при этом обладают некоторыми существенными различиями в реализации недокументированных в стандарте функций.

Одной из самых популярных сред разработки на языке C++ является Microsoft Visual Studio. Бесплатную версию этой среды разработки можно скачать с официального сайта компании Microsoft и использовать без ограничений для обучения программированию. Разработку исходного кода программ для компиляторов Clang и GNU C++ Compiler ведут с помощью самых разных сред разработки, которые чаще всего представляют собой богатые функциональностью текстовые редакторы с подсветкой синтаксиса. К таким редакторам относятся Notepad++, Vim, Emacs, Gedit, Geany и многие другие. Из полноценных сред разработки для этого компилятора можно назвать Code::Blocks, Eclipse, Dev-C++, CLion и многие другие.

Ещё раз нужно отметить, что настоящее издание *не ставит* целью изучение языка C++ и *не позиционируется* как пособие по изучению этого языка. *Нет требования* к выполнению лабораторных работ только на языке C++, но все примеры исходного кода программ для определённости приведены именно на этом языке. Пред-

полагается, что даже если читатель планирует выполнять предложенные задания на языке C++, большую часть информации о синтаксисе и о стандартной библиотеке этого языка ему придётся почерпнуть самостоятельно *из внешних источников*.

Довольно много литературы, подходящей для изучения языка C++, приведено в списке литературы в конце этого издания. Из электронных ресурсов можно отметить сайт http://www.cplusplus.com/, содержащий руководства и интерактивные учебники по языку C++ на английском языке. Информация о реализации языка C++, его сторонней библиотеки и компонентов для него от компании Microsoft можно найти среди другой технической документации в библиотеке MSDN: https://msdn.microsoft.com/ru-ru/. Информацию о реализации языка C++ от проекта GNU можно найти на его официальном сайте https://gcc.gnu.org/ опять же на английском языке. Кроме того, автор рекомендует сайт http://cppreference.com, где в случае необходимости можно получить детальную информацию о той или иной функциональности на уровне стандарта языка.

1.2 Представление целых чисел в памяти компьютера

Целые числа – это наиболее популярный тип данных. Большая часть операций, совершаемых вычислительными устройствами по всему миру, выполняется именно с целыми числами. Для корректной работы с целыми числами следует иметь представление о том, как они хранятся в памяти компьютера.

Каждый тип данных для хранения целых чисел определяет объём памяти, который необходимо выделить для переменной этого типа. Если на хранение одного целого числа выделяется n байт памяти, то этого достаточно, чтобы хранить 2^{8n} различных целых чисел. Конечно, при этом считается, что в одном байте восемь бит.

Для хранения неотрицательных целых чисел достаточно просто хранить цифры их двоичного представления. В этом случае в участке памяти объёмом n байт можно без всяких проблем хранить целые

числа от 0 до $(2^{8n} - 1)$ включительно. Двоичное представление всех этих чисел имеет не более $8n$ значащих цифр. Двоичная запись самого числа 2^{8n} состоит из $(8n + 1)$-й цифры. Целочисленные типы данных, хранящие целые числа таким способом, называются *беззнаковыми*.

В большинстве случае требуется хранить и отрицательные числа тоже. Для этого просто считается, что вторая половина целых чисел, начиная с $2^{8n - 1}$, обозначает не эти самые числа, а соответствующие отрицательные числа в том порядке, в котором они идут, начиная с $(-2^{8n - 1})$. Все эти числа отличаются тем, что в старшем разряде среди имеющихся $8n$ разрядов хранится единица.

Таким образом, для отрицательных целых чисел вместо самого отрицательного числа x хранится двоичное представление числа $(2^{8n} + x)$. Такой способ хранения отрицательных целых чисел называется *дополнительным кодом*. Наименьшее целое число, которое может храниться таким способом, – это число $(-2^{8n - 1})$, а наибольшее число равно $(2^{8n - 1} - 1)$. При таком хранении отрицательных чисел проще реализуются многие операции на аппаратном уровне.

Чтобы записать отрицательное целое число x в дополнительном коде, нужно инвертировать $8n$ младших битов целого числа $(-x)$ и прибавить к полученному числу единицу. Двоичное представление полученного числа и будет дополнительным кодом числа x. Например, $-123_{10} = -01111011_2$, при этом $10000100_2 + 1 = 10000101_2$. Значит, дополнительный код числа –123 в рамках однобайтового целочисленного типа данных со знаком представляет собой 10000101.

Чтобы выяснить, какое число представляется в дополнительном коде в виде двоичной записи числа x, достаточно просто инвертировать все биты этой двоичной записи, прибавить к полученному числу единицу и поставить перед ним знак минус. Например, если инвертировать 10000101 и прибавить единицу, то получится $01111010_2 + 1 = 01111011_2 = 123_{10}$. Значит, запись 10000101 является дополнительным кодом числа –123.

Как видно, алгоритмы перевода числа в дополнительный код и перевода дополнительного кода в число ничем не отличаются. В обоих случаях нужно сделать два действия: 1) инвертировать биты, 2) прибавить единицу. На практике задумываться о том, как именно конкретное число хранится в памяти компьютера, приходится не так уж и часто. Однако это всё же требуется делать при решении некоторых задач, особенно при разработке алгоритмов, содержащих побитовые операции.

В таблице 1 представлены примеры двоичных представлений некоторых целых чисел в рамках восьмибитного целочисленного типа данных со знаком. Именно таким образом эти числа хранятся в памяти компьютера, если для хранения выбран тип данных подобного рода. Ясно, что такой тип данных позволяет хранить целые числа от –128 до +127 включительно.

Таблица 1. Примеры двоичных представлений некоторых целых чисел

Число	Код
−128	10000000
−127	10000001
−100	10011100
−2	11111110
−1	11111111
0	00000000
1	00000001
2	00000010
100	01100100
126	01111110
127	01111111

1.3 Типы данных для хранения целых чисел

Типы данных для хранения целых чисел отличаются размером, выделяемым под хранение одного значения этого типа, а также наличием отдельного способа хранения отрицательных чисел. Типичные размеры целых чисел составляют 1, 2, 4 и 8 байт, и для каждого из таких размеров существует два варианта хранения целых чисел: со

знаком и без знака. Таким образом, существует 8 основных целочисленных типов данных.

В табл. 2 приведены основные типы данных для хранения целых чисел в языке C++. Для каждого типа данных приведено обычное количество памяти, которое занимает значение этого типа, наличие хранимого знака, название этого типа данных в языке C++, количество различных целых чисел, которые можно хранить, минимальное и максимальное хранимые значения, а также количество десятичных цифр. Нужно отметить, что на самом деле в стандарте языка точные размеры для этих типов данных не определены: сказано лишь, что более короткие типы не должны занимать больше памяти, чем более длинные.

Таблица 2. Типы данных для хранения целых чисел

Байт	Знак	Название в C	Кол-во	min	max	Цифр
1	есть	`signed char`	256	−127	128	≈ 3
1	нет	`unsigned char`	256	0	255	≈ 3
2	есть	`short`	65536	−32768	32767	≈ 5
2	нет	`unsigned short`	65536	0	65535	≈ 5
4	есть	`int`	$\approx 4 \cdot 10^9$	$\approx -2 \cdot 10^9$	$\approx 2 \cdot 10^9$	≈ 10
4	нет	`unsigned int`	$\approx 4 \cdot 10^9$	0	$\approx 4 \cdot 10^9$	≈ 10
8	есть	`long long`	$\approx 9 \cdot 10^{18}$	$\approx -9 \cdot 10^{18}$	$\approx 9 \cdot 10^{18}$	≈ 19
8	нет	`unsigned long long`	$\approx 2 \cdot 10^{19}$	0	$\approx 2 \cdot 10^{19}$	≈ 20

Точные размеры и предельные значения различных целочисленных типов данных можно получить внутри программы, подключив заголовочный файл `<limits.h>`. В табл. 3 приведены названия констант из этого заголовочного файла, их описания и примерные значения. Реальные значения могут отличаться в зависимости от аппаратной составляющей вычислительного устройства и от реализации компилятора под него.

В стандарте C++11 определены новые целочисленные типы данных, в том числе типы с гарантированным размером (такие, как int32_t). Кроме того, стандарт C++ определяет класс std::numeric_limits для получения соответствующих констант. Подробную информацию об этом читатель может найти в других источниках.

Таблица 3. Предельные значения из заголовочного файла `limits.h`

Название	Описание	Значение
`CHAR_BIT`	Количество бит в байте	8
`SCHAR_MIN`	Минимум типа `signed char`	–128
`SCHAR_MAX`	Максимум типа `signed char`	127
`UCHAR_MAX`	Максимум типа `unsigned char`	255
`CHAR_MIN`	Минимум типа `char`	SCHAR_MIN
`CHAR_MAX`	Максимум типа `char`	SCHAR_MAX
`SHRT_MIN`	Минимум типа `short int`	-2^{15}
`SHRT_MAX`	Максимум типа `short int`	$2^{15}-1$
`USHRT_MAX`	Максимум типа `unsigned short int`	$2^{16}-1$
`INT_MIN`	Минимум типа `int`	-2^{31}
`INT_MAX`	Максимум типа `int`	$2^{31}-1$
`UINT_MAX`	Максимум типа `unsigned int`	$2^{32}-1$
`LONG_MIN`	Минимум типа `long int`	-2^{31}
`LONG_MAX`	Максимум типа `long int`	$2^{31}-1$
`ULONG_MAX`	Максимум типа `unsigned long int`	$2^{32}-1$
`LLONG_MIN`	Минимум типа `long long`	-2^{63}
`LLONG_MAX`	Максимум типа `long long`	$2^{63}-1$
`ULLONG_MAX`	Максимум типа `unsigned long long`	$2^{64}-1$

На практике для большинства случаев подходит обычный тип данных `int`. Он может хранить целые числа порядка миллиарда, как положительные, так и отрицательные. В случае если этого недостаточно, следует использовать тип данных `long long`. Если и этого недостаточно, приходится использовать специальные алгоритмы для организации работы с большими целыми числами, представление которых не возможно в современных вычислительных устройствах на аппаратном уровне. Технология обработки таких целых чисел называется *длинной арифметикой*.

Практически никогда нет смысла использовать типы данных без знака или типы данных, занимающие меньше четырёх байт, особенно неопытным программистам. Использование более коротких типов позволяет экономить память, что может быть важно лишь в некото-

рых очень редких ситуациях. Типы данных без знака иногда используются для хранения количества элементов, но нужно избегать сравнения значений, одно из которых со знаком, а другое – без знака.

1.4 Операции с целыми числами

В табл. 4 приведены арифметические операции с целыми числами, определённые в языке C++.

Таблица 4. Арифметические операции с целыми числами в C++

Название	Обозначение	Пример	Результат
Сложение	`+`	`7 + 3`	`10`
Вычитание	`-`	`7 - 3`	`4`
Умножение	`*`	`7 * 3`	`21`
Деление нацело	`/`	`7 / 3`	`2`
Остаток от деления	`%`	`7 % 3`	`1`
Унарный плюс	`+`	`+(7)`	`7`
Унарный минус	`-`	`-(7)`	`-7`
Префиксный инкремент	`++`	`++a`	`a + 1`
Постфиксный инкремент	`++`	`a++`	`a`
Префиксный декремент	`--`	`--a`	`a - 1`
Постфиксный декремент	`--`	`a--`	`a`

Пять основных бинарных операций: сложение, вычитание, умножение, целочисленное деление и остаток от деления. Следует с осторожностью применять последние две операции деления к отрицательным числам: они работают не совсем так, как многие начинающие программисты подразумевают интуитивно. Рекомендуется выполнять необходимые проверки на наличие отрицательных операндов перед выполнением таких операций, по крайней мере, пока язык C++ ещё не достаточно хорошо изучен. Имеются также унарные операции плюс и минус, первая из которых практически не используется.

Операции инкремента и декремента используются для увеличения и уменьшения значения некоторой переменной на единицу соответственно. Префиксная версия этих операций сначала выполняет операцию инкремента либо декремента, а затем возвращает резуль-

тат в выражение. Постфиксная версия этих операций, напротив, сначала возвращает ещё не изменённое значение, а затем выполняет соответствующую операцию. Использование этих операций позволяет в некоторых случаях писать довольно элегантный код, но в то же время снижает его читабельность.

В табл. 5 приведены примеры бинарных операций сравнения для целых чисел.

Таблица 5. Операции сравнения целых чисел в C++

Название	Обозначение	Пример	Результат
Равенство	`==`	`7 == 3`	`false`
Неравенство	`!=`	`7 != 3`	`true`
Больше	`>`	`7 > 3`	`true`
Меньше	`<`	`7 < 3`	`false`
Больше либо равно	`>=`	`7 >= 3`	`true`
Меньше либо равно	`<=`	`7 <= 3`	`false`

С этими операциями не должно возникнуть никаких проблем: их принцип работы интуитивно понятен. Эти же операции могут применяться не только для сравнения целых чисел, но и вообще для сравнения значений любого типа. Начинающим следует обратить особое внимание, что сравнение на равенство выполняется в языке C++ операцией ==. Во многих выражениях (например, в условном операторе) использование одного знака равенства синтаксически верно, но будет соответствовать присваиванию.

В табл. 6 представлены побитовые операции с целыми числами. Это аналоги соответствующих логических операций, выполняющихся с каждым соответствующим битом операндов. Операции битового сдвига сдвигают двоичное представление числа в заданную сторону на количество разрядов, равное правому операнду. Неопытные программисты редко используют побитовые операции, поскольку их использование требует хорошего понимания того, как целые числа хранятся в памяти компьютера. Однако эти операции выполняются очень быстро и могут существенно ускорить работу программы в

некоторых специальных случаях. Например, операции битового сдвига могут использоваться как аналоги умножения и деления на степени двойки.

Таблица 6. Побитовые операции в C++

Название	Обозначение	Пример	Результат
Инверсия битов	~	~7	-8
Побитовая конъюнкция	&	7 & 3	3
Побитовая дизъюнкция	|	7 | 3	7
Побитовое сложение по модулю 2	^	7 ^ 3	4
Битовый сдвиг влево	<<	7 << 3	56
Битовый сдвиг вправо	>>	7 >> 3	0

В табл. 7 приведены операции присваивания, определённые для целочисленных переменных.

Таблица 7. Операции присваивания для целых чисел в C++

Название	Обозначение	Пример	Результат
Присваивание	=	a = 3	a
Присваивание со сложением	+=	a += 3	a + 3
Присваивание с вычитанием	-=	a -= 3	a - 3
Присваивание с умножением	*=	a *= 3	a * 3
Присваивание с делением	/=	a /= 3	a / 3
Присваивание с остатком	%=	a %= 3	a % 3
Присваивание с конъюнкцией	&=	a &= 3	a & 3
Присваивание с дизъюнкцией	|=	a |= 3	a | 3
Присваивание битовым сложением	^=	a ^= 3	a ^ 3
Присваивание со сдвигом влево	<<=	a <<= 3	a << 3
Присваивание со сдвигом вправо	>>=	a >>= 3	a >> 3

Левым операндом такой операции должна выступать переменная или другой программный объект, имеющий свой собственный адрес в памяти. Чаще всего используется обычная операция присваивания. Остальные операции работают по одному и тому же принципу: сначала с левым и правым операндом выполняется операция, а результат записывается обратно в левый операнд. Все операции присваива-

ния возвращают ссылку на левый операнд, так что можно использовать это в более сложных выражениях.

Существует ряд других операций, некоторые из которых могут быть использованы и для целых чисел. Например, операции приведения типов используются для преобразования значений одного типа к значениям другого типа. Классическая запись оператора приведения типа выглядит как «(тип)выражение» и приводит значение выражения к заданному типу. Например, «`(long long)0`» возвращает `0` в рамках типа данных `long long`. Все типы данных, хранящие целые числа, могут легко приводиться друг к другу таким способом.

1.5 Целочисленное переполнение

Главная проблема, которая может возникнуть при работе с целыми числами, – это целочисленное переполнение. Эта проблема возникает из-за ограниченности участка памяти, отведённого под хранение значения конкретного типа данных. При выполнении операций с целыми числами, главным образом арифметических, результат операции может не помещаться в область памяти заданного размера, то есть выходить за границы диапазона значений, допустимых для заданного типа данных, даже если операнды легко умещались в рамки этого типа. Такая ситуация и называется *целочисленным переполнением*.

В результате целочисленного переполнения сохраняются только младшие разряды значения выражения, а старшие разряды, не влезающие в область памяти заданного размера, просто отбрасываются. Это единственный случай, когда в результате операции с целыми числами будет получен неправильный результат. Следует всегда помнить об этой проблеме при написании исходного кода программы и стараться всячески избегать целочисленного переполнения.

В листинге 3 приведён пример исходного кода программы на C++, демонстрирующей целочисленное переполнение.

Листинг 3. Пример целочисленного переполнения на C++

```
#include <iostream>

int main()
{
	std::cout << 2000000000 + 2000000000 << std::endl;
	std::cout << 1000000 * 1000000 << std::endl;
	return 0;
}
```

Все целочисленные литералы, вроде «2000000000» или «1000000» по умолчанию считаются принадлежащими типу данных `int`. Если это 32-битное целое число со знаком, то в результате выполнения программы будут выведены числа «-294967296» и «-727379968» соответственно. Это связано с тем, что $2{\times}10^9 + 2{\times}10^9 = 4{\times}10^9 > 2^{31} - 1$ и $10^6 \times 10^6 = 10^{12} > 2^{31} - 1$. Формально же результат таких операций не определён и может привести к любому непредвиденному поведению программы в зависимости от реализации компилятора и архитектуры процессора.

Самый простой способ избежать целочисленного переполнения – использовать более широкий тип данных для тех участков программы, в которых могут возникнуть настолько большие числа. Например, все те же самые операции из листинга 3 не привели бы к целочисленному переполнению, если бы выполнялись в рамках 64-битного типа данных `long long`. Но и этот тип данных позволяет хранить числа лишь порядка 10^{18}.

Для того чтобы явно указать, что целочисленный литерал должен относиться к типу данных `long long`, требуется дописать к нему суффикс «LL». Например, операции «2000000000LL + 2000000000LL» и «1000000LL * 1000000LL» выполнятся корректно. Другой вариант – сохранить значения операндов в переменные типа `long long`. Более того, достаточно, чтобы хотя бы один операнд был типа `long long`, поскольку при выполнении бинарных арифметических операций результат приводится к самому широкому из типов данных операндов.

Листинг 4. Другой пример целочисленного переполнения на C++

```
#include <iostream>

int main()
{
	long long a = 2000000000 + 2000000000;
	long long b = 1000000 * 1000000;
	std::cout << a << std::endl;
	std::cout << b << std::endl;
	return 0;
}
```

При этом следует отдельно отметить, что сохранение результата в переменную более широкого типа не решает проблему, поскольку переполнение уже произошло при выполнении операции и результат, хотя и приведётся к более широкому типу, уже будет неправильным. В листинге 4 приведён такой пример. Следует ещё раз отметить, что это пример *неправильного* решения проблемы, так поступать *нельзя*. Результат выполнения листинга 4 ничем не отличается от результата выполнения листинга 3.

1.6 Задание на лабораторную работу № 1

Вариант 0. A+B

Программа должна складывать два целых числа.

Входные данные. Два целых числа.

Выходные данные. Сумма этих чисел.

Пример текстового интерфейса пользователя

```
A+B
Input an integer>2
Input an integer>3
Result: 5
Continue? (Y/N)>Y
Input an integer>2
Input an integer>2
Result: 4
Continue? (Y/N)>N
```

Вариант 1. Из десятичной системы в двоичную

Программа должна переводить целое число из десятичной системы счисления в двоичную.

Входные данные. Целое число в десятичной системе счисления.

Выходные данные. То же целое число в двоичной системе счисления.

Пример текстового интерфейса пользователя

```
Translating a number from decimal to binary
Input an integer>123
Result: 1111011
Continue? (Y/N)>Y
Input an integer>1234
Result: 10011010010
Continue? (Y/N)>N
```

Вариант 2. Из десятичной системы в шестнадцатеричную

Программа должна переводить целое число из десятичной системы счисления в шестнадцатеричную.

Входные данные. Целое число в десятичной системе счисления.

Выходные данные. То же целое число в шестнадцатеричной системе счисления.

Пример текстового интерфейса пользователя

```
Translating a number from decimal to hexadecimal
Input an integer>123
Result: 7B
Continue? (Y/N)>Y
Input an integer>1234
Result: 4D2
Continue? (Y/N)>N
```

Вариант 3. Из двоичной системы в десятичную

Программа должна переводить целое число из двоичной системы счисления в десятичную.

Входные данные. Целое число в двоичной системе счисления.

Выходные данные. То же целое число в десятичной системе счисления.

Пример текстового интерфейса пользователя

```
Translating a number from binary to decimal
Input a binary integer>1111011
Result: 123
Continue? (Y/N)>Y
Input an integer>10011010010
Result: 1234
Continue? (Y/N)>N
```

Вариант 4. Из шестнадцатеричной системы в десятичную

Программа должна переводить целое число из шестнадцатеричной системы счисления в десятичную.

Входные данные. Целое число в шестнадцатеричной системе счисления.

Выходные данные. То же целое число в десятичной системе счисления.

Пример текстового интерфейса пользователя

```
Translating a number from hexadecimal to decimal
Input a binary integer>7B
Result: 123
Continue? (Y/N)>Y
Input an integer>4D2
Result: 1234
Continue? (Y/N)>N
```

Вариант 5. Двоичная система с избытком цифр

Рассмотрим двоичную систему счисления с избытком цифр: в такой системе счисления могут использоваться все 10 цифр, но основанием системы счисления всё равно является число 2. Программа должна сравнивать два целых числа, записанные в такой системе счисления, и определять, какое из них больше, либо сообщать о том, что они равны.

Входные данные. Два целых числа в двоичной системе счисления с избытком цифр.

Выходные данные. Сообщение о том, какое из этих двух чисел больше, либо сообщение об их равенстве.

Пример текстового интерфейса пользователя

```
Comparison of integers in a binary system with an
excess of digits
Input an integer>23
Input an integer>16
23 < 16
Continue? (Y/N)>Y
Input an integer>45
Input an integer>37
45 = 37
Continue? (Y/N)>N
```

Вариант 6. Дробь

Программа должна для заданного натурального числа *n* выводить первые *m* цифр после запятой в десятичной записи дроби 1 / *n*.

Входные данные. Целые положительные числа *n* и *m*.

Выходные данные. Первые *m* цифр после запятой в десятичной записи дроби 1 / *n*.

Пример текстового интерфейса пользователя

```
Digits of fraction 1 / n
Input a positive integer n>7
Input the number of digits>6
0.142857
Continue? (Y/N)>Y
Input a positive integer n>2
Input the number of digits>20
0.50000000000000000000
Continue? (Y/N)>N
```

Вариант 7. Проверка на степень двойки

Программа должна проверять, является ли заданное натуральное число натуральной степенью двойки.

Входные данные. Целое положительное число.

Выходные данные. Сообщение о том, является введённое число некоторой целой положительной степенью двойки или нет.

Пример текстового интерфейса пользователя

```
Check for a power of two
Input a positive integer>8
8 is a power of 2
Continue? (Y/N)>Y
Input a positive integer>9
9 is not a power of 2
Continue? (Y/N)>N
```

Вариант 8. Високосные года

Год является високосным, если его номер делится на 4. Но если он делится на 100, то не является високосным, если только он не делится ещё и на 400. Программа должна определять количество високосных годов на заданном пользователем временном отрезке.

Входные данные. Два положительных целых числа a и b ($a < b$) – номера первого и последнего годов во временном отрезке.

Выходные данные. Количество високосных годов, начиная с года a и заканчивая годом b включительно.

Пример текстового интерфейса пользователя

```
The number of leap years on a time interval
Input a start year>1900
Input a final year>2000
The number of leap years: 25
Continue? (Y/N)>Y
Input a start year>2000
Input a final year>2010
The number of leap years: 3
Continue? (Y/N)>N
```

Вариант 9. Форматирование времени

Программа должна переводить количество секунд, прошедших с начала суток, в формат «часы:минуты:секунды».

Входные данные. Целое неотрицательное число – количество секунд, прошедших с начала суток.

Выходные данные. Количество часов, прошедших с начала суток, количество минут, прошедших с начала последнего часа, и количество секунд, прошедших с начала последней минуты, в формате «часы:минуты:секунды».

Пример текстового интерфейса пользователя

```
Time formatter
Input the number of seconds>43200
12:00:00
Continue? (Y/N)>Y
Input the number of seconds>10000
02:46:40
Continue? (Y/N)>N
```

Вариант 10. Дискретное извлечение корня

Дискретным корнем из натурального числа n по модулю m называется такое наименьшее натуральное число k, что $k^2 \equiv n \pmod{m}$, то есть такое, что число $(k^2 - n)$ делится на m без остатка. Программа должна извлекать дискретный корень из заданного натурального числа по заданному простому модулю, если это возможно.

Входные данные. Целое положительное число n и простое число m.

Выходные данные. Дискретный корень из n по модулю m, либо сообщение о том, что его не существует.

Пример текстового интерфейса пользователя

```
Discrete Square Root
Input a positive integer>13
Input a modulo>17
Result: 8
Continue? (Y/N)>Y
Input a positive integer>3
Input a modulo>11
Result: 5
Continue? (Y/N)>N
```

Вариант 11. Дискретное логарифмирование

Дискретным логарифмом натурального числа n по модулю m называется такое наименьшее целое неотрицательное число k, что $2^k \equiv n \pmod{m}$, то есть такое, что число $(2^k - n)$ делится на m без остатка. Программа должна вычислять дискретный логарифм заданного натурального числа по заданному простому модулю, если это возможно.

Входные данные. Целое положительное число n и простое число m.

Выходные данные. Дискретный логарифм n по модулю m, либо сообщение о том, что его не существует.

Пример текстового интерфейса пользователя

```
Discrete logarithm
Input a positive integer>13
Input a modulo>17
Result: 6
Continue? (Y/N)>Y
Input a positive integer>22
Input a modulo>11
Impossible
Continue? (Y/N)>N
```

Вариант 12. Обратный элемент в кольце по модулю

Обратный элемент к натуральному числу n по модулю m – это такое наименьшее натуральное число k, что $n \times k \equiv 1 \pmod{m}$, то есть такое, что число $(n \times k - 1)$ делится на m без остатка. Программа должна находить обратный элемент к заданному числу по заданному модулю, если он существует.

Входные данные. Целые положительные числа n и m.

Выходные данные. Обратный элемент к числу n по модулю m, либо сообщение о том, что его не существует.

Пример текстового интерфейса пользователя

```
Inverse element
Input a positive integer>13
Input a modulo>17
Result: 4
Continue? (Y/N)>Y
Input a positive integer>36
Input a modulo>100
Impossible
Continue? (Y/N)>N
```

Вариант 13. Цифровой корень числа

Цифровым корнем целого неотрицательного числа называется цифра, являющаяся результатом последовательной замены текущего числа суммой его цифр. Программа должна вычислять цифровой корень заданного целого неотрицательного числа.

Входные данные. Целое неотрицательное число.

Выходные данные. Цифра, являющаяся цифровым корнем этого числа.

Пример текстового интерфейса пользователя

```
Digital root
Input a nonnegative integer>678
Result: 3
Continue? (Y/N)>Y
Input a nonnegative integer>5678
Result: 8
Continue? (Y/N)>N
```

Вариант 14. Наибольший общий делитель

Наибольший общий делитель двух целых чисел – это наибольшее целое число, на которое оба этих числа делятся без остатка. Программа должна находить наибольший общий делитель двух целых чисел.

Входные данные. Два целых числа.

Выходные данные. Наибольший общий делитель этих чисел.

Пример текстового интерфейса пользователя

```
Greatest common divisor
Input an integer>12
Input an integer>15
Result: 3
Continue? (Y/N)>Y
Input an integer>21
Input an integer>10
Result: 1
Continue? (Y/N)>N
```

Вариант 15. Функция Эйлера

Функция Эйлера для натурального числа *n* принимает значение, равное количеству натуральных чисел, не превосходящих *n* и не имеющих с *n* общих делителей, отличных от единицы. Требуется написать программу, вычисляющую значение функции Эйлера для введённого пользователем натурального аргумента.

Входные данные. Целое положительное число *n*.

Выходные данные. Значение функции Эйлера в точке *n*.

Пример текстового интерфейса пользователя

```
Euler's totient function
Input a positive integer>12
Result: 4
Continue? (Y/N)>Y
Input a positive integer>60
Result: 16
Continue? (Y/N)>N
```

Вариант 16. Сложное число

Будем считать, что число тем сложнее, чем больше у него натуральных делителей. Программа должна находить самое сложное число, не превосходящее заданное пользователем число. Если существу-

ет несколько чисел с одинаково большим количеством делителей, то программа должна находить наименьшее из них.

Входные данные. Целое положительное число *n*.

Выходные данные. Самое сложное натуральное число, не превосходящее *n*.

Пример текстового интерфейса пользователя

```
Highly composite number
Input a positive integer>100
Result: 60
Continue? (Y/N)>Y
Input a positive integer>1000
Result: 840
Continue? (Y/N)>N
```

Вариант 17. Числа Армстронга

Числа Армстронга – это натуральные числа, равные сумме своих цифр, возведённых в степень количества цифр в их записи. Программа должна находить все числа Армстронга на заданном пользователем отрезке.

Входные данные. Два положительных целых числа *a* и *b* ($a \leq b$) – границы отрезка.

Выходные данные. Все числа Армстронга, не меньшие *a* и не большие *b*.

Пример текстового интерфейса пользователя

```
Armstrong numbers
Input a left border>100
Input a right border>1000
Armstrong numbers: 153, 370, 371, 407
Continue? (Y/N)>Y
Input a left border>1000
Input a right border>10000
Armstrong numbers: 1634, 8208, 9474
Continue? (Y/N)>N
```

Вариант 18. Счастливые билетики

Проездной билет называется счастливым, если сумма цифр первой половины десятичной записи его номера равна сумме цифр второй половины этой записи. Программа должна определять, сколько существует счастливых проездных билетов, номер которых состоит из заданного пользователем чётного количества десятичных цифр. Номера билетиков – это целые неотрицательные числа.

Входные данные. Чётное положительное целое число *n*.

Выходные данные. Количество счастливых билетиков, в номерах которых *n* цифр.

Пример текстового интерфейса пользователя

```
Lucky tickets
Input the number of digits>4
The number of 4-digit lucky tickets: 670
Continue? (Y/N)>Y
Input the number of digits>6
The number of 4-digit lucky tickets: 55252
Continue? (Y/N)>N
```

Вариант 19. Проблема 196 и числа Лишрел

Программа должна определять, через какое количество последовательных замен текущего целого числа на сумму этого числа и числа, полученного переворотом десятичной записи текущего числа задом наперёд, получится число-палиндром, то есть число, читающееся одинаково как слева направо, так и справа налево. Также программа должна выводить само получившееся число-палиндром.

Входные данные. Неотрицательное целое число.

Выходные данные. Количество последовательных замен исходного числа на сумму этого числа и перевёрнутого числа до получения палиндрома и сам получившийся палиндром.

Пример текстового интерфейса пользователя

```
196-algorithm
Input a nonnegative integer>59
3 iterations to get 1111
Continue? (Y/N)>Y
Input a nonnegative integer>89
24 iterations to get 8813200023188
Continue? (Y/N)>N
```

Вариант 20. Числа-градины и гипотеза Сиракуз

Рассмотрим последовательность чисел, начинающуюся с некоторого заданного натурального числа, каждое последующее число в которой получается из предыдущего делением пополам, если предыдущее число было чётным, либо умножением на три и прибавлением единицы, если это было не так. Гипотеза Коллатца состоит в том, что для любого начального натурального числа эта последовательность приходит к единице. Программа должна проверять эту гипотезу для заданного натурального числа и выводить всю последовательность, начиная с заданного числа и заканчивая единицей.

Входные данные. Положительное целое число.

Выходные данные. Описанная последовательность чисел, начинающаяся с этого числа и заканчивающаяся единицей.

Пример текстового интерфейса пользователя

```
Collatz conjecture check
Input a positive integer>12
12 -> 6 -> 3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
Continue? (Y/N)>Y
Input a positive integer>20
20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1
Continue? (Y/N)>N
```

Вариант 21. Гипотеза Гольдбаха

Гипотеза Гольдбаха, не доказанная и не опровергнутая на протяжении уже почти трёх столетий, заключается в том, что любое чётное число, большее двух, представимо в виде суммы двух простых чисел. Программа должна проверять эту гипотезу для введённого пользователем чётного числа, то есть указывать два простых числа, которые в сумме дают заданное чётное число. Простое число – это натуральное число, имеющее ровно два различных натуральных делителя.

Входные данные. Положительное чётное целое число.

Выходные данные. Два простых числа, дающих в сумме это число.

Пример текстового интерфейса пользователя

```
Goldbach's conjecture
Input a positive even integer>12
12 = 5 + 7
Continue? (Y/N)>Y
Input a positive even integer>30
30 = 7 + 23
Continue? (Y/N)>N
```

Вариант 22. Скатерть Улама и треугольник Клаубера

Программа должна определять, сколько простых чисел содержится среди заданного количества начальных элементов последовательности $a_n = n\ (n - 1) + k$ для заданного k. Простое число – это натуральное число, имеющее ровно два различных натуральных делителя. Элементы последовательности нумеруются с единицы.

Входные данные. Целое положительное число k и количество элементов последовательности, которые нужно проверить.

Выходные данные. Количество элементов последовательности, являющихся простыми числами.

Пример текстового интерфейса пользователя

```
Primality test for elements a(n) = n (n - 1) + k
Input a positive integer k>5
Input the number of elements>10
The number of primes: 6
Continue? (Y/N)>Y
Input a positive integer k>11
Input the number of elements>10
The number of primes: 10
Continue? (Y/N)>N
```

Вариант 23. Пёстрые числа

Натуральное число называется пёстрым, если все цифры в его десятичной записи различны. Программа должна определять количество пёстрых чисел на заданном пользователем отрезке.

Входные данные. Два положительных целых числа a и b ($a \le b$) – границы отрезка.

Выходные данные. Количество пёстрых чисел, не меньших a и не больших b.

Пример текстового интерфейса пользователя

```
Colorful numbers
Input a left border>10
Input a right border>20
The number of colorful numbers: 10
Continue? (Y/N)>Y
Input a left border>100
Input a right border>200
The number of colorful numbers: 72
Continue? (Y/N)>N
```

Вариант 24. Совершенные числа

Натуральное число называется *совершенным*, если оно равно сумме всех своих делителей, отличных от него самого. Первые 4 совершенных числа были указаны в работе древнегреческого философа Никомаха Герасского во II веке нашей эры, а пятое число было обна-

ружено лишь спустя 13 столетий. Программа должна отыскивать все совершенные числа, не превышающие заданное число.

Входные данные. Целое положительное число *n*.

Выходные данные. Все совершенные числа, не превосходящее *n*.

Пример текстового интерфейса пользователя

```
Perfect numbers
Input a positive integer>100
Perfect numbers: 6, 28
Continue? (Y/N)>Y
Input a positive integer>10000
Perfect numbers: 6, 28, 496, 8128
Continue? (Y/N)>N
```

Вариант 25. Автоморфные числа

Натуральное число называется *автоморфным*, если десятичная запись его квадрата оканчивается на десятичную запись этого числа. Требуется написать программу для отыскания всех автоморфных чисел на заданном пользователем отрезке.

Входные данные. Два положительных целых числа *a* и *b* ($a \le b$) – границы отрезка.

Выходные данные. Все автоморфные числа, не меньшие *a* и не большие *b*.

Пример текстового интерфейса пользователя

```
Automorphic numbers
Input a left border>10
Input a right border>100
Automorphic numbers: 25, 76
Continue? (Y/N)>Y
Input a left border>100
Input a right border>1000
Automorphic numbers: 376, 625
Continue? (Y/N)>N
```

Вариант 26. Пифагоровы тройки

Три натуральных числа называются *пифагоровой тройкой*, если существует прямоугольный треугольник, длины сторон которого равны этим числам. Требуется написать программу для поиска заданного пользователем количества пифагоровых троек, наибольший общий делитель чисел каждой из которых равен единице.

Входные данные. Положительное целое число – количество пифагоровых троек.

Выходные данные. Заданное количество различных пифагоровых троек, наибольший общий делитель каждой из которых равен единице.

Пример текстового интерфейса пользователя

```
Pythagorean triples
Input the number of Pythagorean triples>2
(3, 4, 5), (5, 12, 13)
Continue? (Y/N)>Y
Input the number of Pythagorean triples>3
(3, 4, 5), (5, 12, 13), (8, 15, 17)
Continue? (Y/N)>N
```

Вариант 27. Факторизация числа

Программа должна разбивать заданное натуральное число на простые множители. Простое число – это число, у которого ровно два различных натуральных делителя.

Входные данные. Положительное целое число.

Выходные данные. Разбиение заданного числа на простые делители.

Пример текстового интерфейса пользователя

```
Factorization
Input a positive integer>12
12 = 2 * 2 * 3
Continue? (Y/N)>Y
Input a positive integer>225
225 = 3 * 3 * 5 * 5
Continue? (Y/N)>N
```

Вариант 28. Калькулятор целых чисел

Состоянием программы является целое число. Пользователь может выполнять с этим числом арифметические операции: сложение, вычитание, умножение и целочисленное деление, изменяя состояние. Программа должна визуализировать результат каждой операции.

Входные данные. Операции и целые операнды.

Выходные данные. Результат каждой операции.

Пример текстового интерфейса пользователя

```
Integer calculator
Initial state: 0
Input an operation (+/-/*//)>+
Input an operand>7
Result: 7
Continue? (Y/N)>Y
Input an operation (+/-/*//)>/
Input an operand>3
Result: 2
Continue? (Y/N)>N
```

Вариант 29. Игра со спичками

Игра со спичками заключается в следующем. На столе лежат n спичек, играют двое, ходят по очереди. За свой ход игрок может взять от a до b спичек включительно. Проигрывает тот, кто не может сделать ход. Требуется написать программу, играющую с пользователем в эту игру. Перед началом игры пользователь задаёт числа a, b и n, после чего пользователь и программа по очереди делают ходы. В конце игры программа определяет и объявляет победителя.

Входные данные. Параметры игры a, b и n ($a \le b$), а также ходы игрока.

Выходные данные. Результат каждого хода и общий результат игры.

Пример текстового интерфейса пользователя

```
Game with matches
Input the total number of matches>100
Input the minimal number of taken matches>20
Input the maximal number of taken matches>30
Current number of matches: 100
Your turn>30
Current number of matches: 70
Bot takes 30 matches
Current number of matches: 40
Your turn>20
Current number of matches: 20
Bot takes 20 matches
Current number of matches: 0
You lose! Game over.
```

Вариант 30. Мастер угадывания чисел

Программа должна угадывать целое число, задуманное пользователем. Программа может предпринимать попытки угадать задуманное число, предлагая пользователю некоторое предполагаемое целое число, на что пользователь должен ответить, больше его число, чем предполагаемое программой, меньше его, или же программа угадала задуманное число. Программа может предпринимать такие попытки, пока не угадает задуманное число.

Входные данные. Ответы на предположения программы: больше, меньше или равно.

Выходные данные. Предположения о задуманном числе.

Пример текстового интерфейса пользователя

```
Number guessing wizard
Guess: 1
Input answer (is your number >, < or =)>
>
Guess: 2
```

```
Input answer (is your number >, < or =)>
>
Guess: 4
Input answer (is your number >, < or =)>
>
Guess: 8
Input answer (is your number >, < or =)>
>
Guess: 16
Input answer (is your number >, < or =)>
<
Guess: 13
Input answer (is your number >, < or =)>
=
Game over
```

1.7 Контрольные вопросы

1. Как целые числа хранятся в памяти компьютера?

2. Чем отличаются способы хранения целых чисел со знаком и без знака?

3. Что такое дополнительный код и как получить дополнительный код заданного отрицательного числа?

4. Какие типы данных существуют для хранения целых чисел? Чем они отличаются?

5. Какие типы данных чаще всего используются для хранения целых чисел? Какого порядка числа они хранят?

6. Что такое приведение типов?

7. Какие арифметические операции с целыми числами поддерживаются вычислительными устройствами?

8. Какие побитовые операции с целыми числами поддерживаются вычислительными устройствами?

9. Для чего могут использоваться операции битового сдвига?

10. Что такое целочисленное переполнение и как его избежать?

1.8 Пример выполнения лабораторной работы

Для выполнения любой лабораторной работы требуется написать исходный код программы. Это можно делать на любом языке программирования и в любой среде разработки. Например, можно использовать язык C++ и среду разработки Microsoft Visual Studio Community Edition, которую можно бесплатно скачать с официального сайта Microsoft.

В случае использования Microsoft Visual Studio при разработке рекомендуется создавать пустой проект, как показано на рис. 2. После этого будет создан пустой проект, в котором нет ни одного файла. Можно добавить туда файл с исходным кодом на C++, например, щёлкнув правой кнопкой мыши по проекту в обозревателе решений и выбрав «Добавить» – «Создать элемент» – «Файл C++». Окно, возникающее при этом, показано на рис. 3. После этого в проекте будет единственный пустой файл с расширением «cpp», в котором можно писать исходный код программы. Не рекомендуется использовать шаблоны проектов, в которых изначально содержатся файлы с исходным кодом, если отсутствует понимание принципов работы и назначения участков исходного кода в этих файлах.

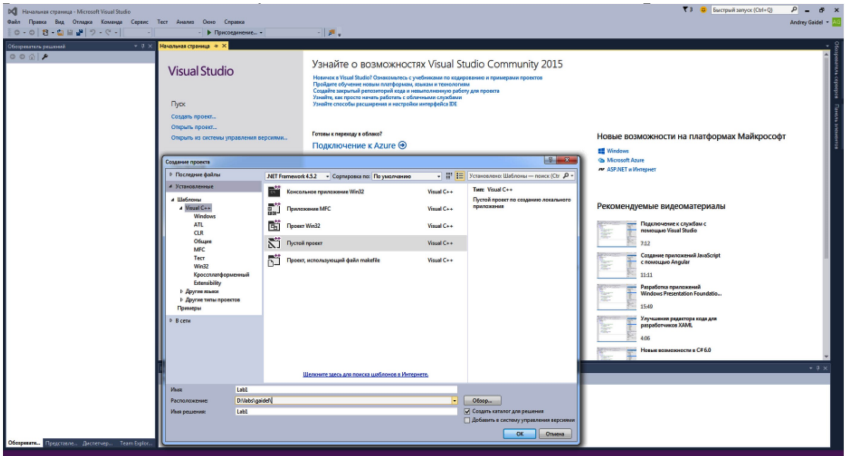

Рис. 2. Создание проекта в Visual Studio

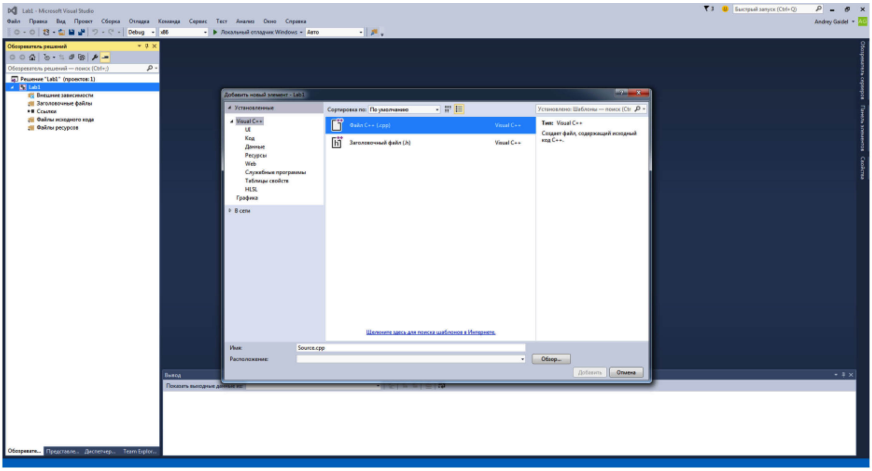

Рис. 3. Добавление файла в проект

В листинге 5 приведён в качестве примера исходный код программы, являющейся возможным результатом выполнения нулевого варианта первой лабораторной работы. Большая часть кода посвящена тонкостям работы с потоками с целью организации безопасного ввода: программа не должна некорректно завершать свою работу в случае ввода пользователем некорректных данных. Вместо этого она должна выводить соответствующее сообщение и по возможности просить повторить ввод.

Листинг 5. Пример исходного кода выполненной лабораторной работы

```
#include <iostream>
#include <string>

const char EOLN = '\n';
const char YES_CHAR = 'Y';
const char NO_CHAR = 'N';
const int LEFT_BOUND = -1000000000;
const int RIGHT_BOUND = +1000000000;
const std::string ABOUT_MESSAGE = "A+B";
const std::string CONTINUE_MESSAGE = "Continue? (Y/N)>";
const std::string INCORRECT_MESSAGE =
"Input is incorrect. Try again>";
const std::string INPUT_MESSAGE = "Input an integer>";
```

```
const std::string OUT_OF_BOUNDS_MESSAGE =
"This number is out of bounds";
const std::string OUTPUT_MESSAGE = "Result: ";
const std::string SKIP_CHARACTERS = " ";

void ClearInputStream(std::istream &in)
{
    in.clear();
    while (in.peek() != EOLN && in.peek() != EOF)
    {
        in.get();
    }
}

int Seek(std::istream &in)
{
    while (in.peek() != EOLN &&
SKIP_CHARACTERS.find((char)in.peek()) != std::string::npos)
    {
        in.get();
    }
    return in.peek();
}

bool CheckBounds(int n)
{
    bool ok = (LEFT_BOUND <= n && n <= RIGHT_BOUND);
    if (!ok)
    {
        std::cout << OUT_OF_BOUNDS_MESSAGE << " [" <<
LEFT_BOUND << ", " << RIGHT_BOUND << "]" << std::endl;
    }
    return ok;
}

int ReadInt(std::istream &in)
{
    std::cout << INPUT_MESSAGE;
    int ans;
    in >> ans;
```

```
    while (!in || Seek(in) != EOLN || !CheckBounds(ans))
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    return ans;
}

bool NeedContinue(std::istream &in)
{
    std::cout << CONTINUE_MESSAGE;
    char ans;
    in >> ans;
    while (!in || Seek(in) != EOLN || ans != YES_CHAR && ans
!= NO_CHAR)
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    return ans == YES_CHAR;
}

int main()
{
    std::cout << ABOUT_MESSAGE << std::endl;
    bool cont = true;
    while (cont)
    {
        int a = ReadInt(std::cin);
        int b = ReadInt(std::cin);
        std::cout << OUTPUT_MESSAGE << a + b << std::endl;
        cont = NeedContinue(std::cin);
    }
    return 0;
}
```

Следует отметить, что организация защиты программы от некорректного ввода *не является* обязательным требованием для успешной сдачи лабораторной работы, а лишь влияет на оценку в случае успешной сдачи. Начинающему программисту, не знакомому детально с принципами работы потоков ввода-вывода, может быть очень сложно решить задачу организации безопасного ввода, но в этом случае и не стоит этого делать. В сущности, для успешной сдачи нулевого варианта первой лабораторной работы достаточно и программы, приведённой в листинге 2. Эта программа состоит из девяти строчек, а не из восьмидесяти пяти, и использует только простые операции. Её написание не представляет труда даже для новичка. Для допуска к экзамену достаточно просто сдать лабораторные работы на любую оценку, так что имеет смысл соблюдать баланс между желаемой оценкой и временем, которое реально может быть потрачено на выполнение лабораторной работы.

2 Числа с плавающей запятой

2.1 Хранение чисел с плавающей запятой в памяти

Часто при написании программы возникает необходимость работы с дробными числами. Для этого можно использовать, например, пару целых чисел для хранения числителя и знаменателя некоторого рационального числа. В этом случае операции с такими числами придётся реализовывать самостоятельно.

Большинство вычислительных устройств поддерживают хранение дробных чисел в виде чисел с плавающей запятой. При этом хранится знак числа, несколько первых старших разрядов записи числа в двоичной системе счисления, а также положение запятой в записи этого числа. Всё это должно храниться в одном участке памяти. Схема разбиения блока памяти на эти три части представлена на рис. 4.

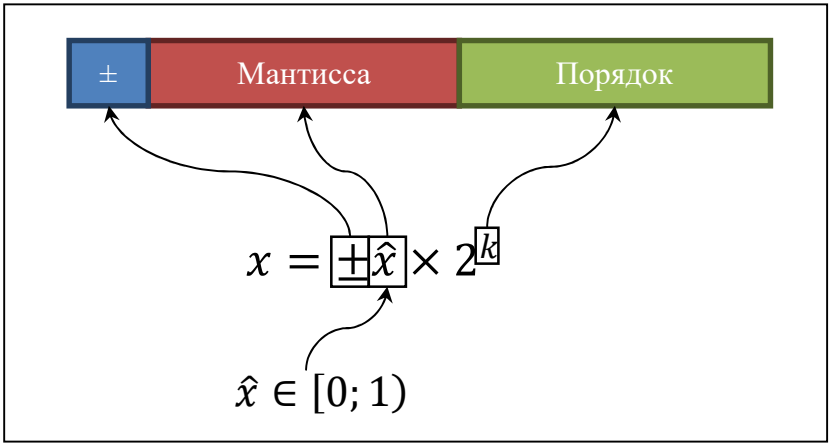

Рис. 4. Представление числа с плавающей запятой в памяти компьютера

Любое вещественное число x, кроме нуля, может быть представлено в виде

$$x = (-1)^s \cdot \hat{x} \cdot 2^k,$$

где $s \in \{0;1\}$ – знак числа, $\hat{x} \in [0{,}5;1)$ – значащие цифры числа, k – порядок числа. По сути $\hat{x}$ – это вещественное число, двоичная запись которого начинается на «0,1». Именно такое представление числа используется при хранении его в виде числа с плавающей запятой.

На знак отводится 1 бит памяти. Мантисса хранит непосредственно значащие цифры числа, начиная со старших разрядов. Если младшие разряды не помещаются в мантиссу, они отбрасываются. Порядок – это целое число, обозначающее положение запятой в двоичной записи указанного числа. Оно хранится, как обычное целое число. В том числе, если порядок должен быть отрицательным, он хранится в дополнительном коде.

Понятно, что при таком хранении возможно неоднозначное представление некоторых чисел. Так, например, присутствуют два нуля: положительный и отрицательный. Они отличаются знаком, а все остальные биты просто заполняются нулями. По идее при сравнении положительного и отрицательного нуля на точное равенство должна возвращаться истина, но на практике это не всегда так.

Для решения этой проблемы есть подход, при котором первая значащая единица в двоичной записи числа не хранится. В двоичной системе счисления только две цифры: 0 и 1, так что первая значащая цифра всегда равна единице. Исключение составляет только число 0, в двоичной записи которого вообще не содержится ни одной единицы. При таком хранении большинство чисел хранится однозначно, но ноль больше не может храниться обычным способом. Участок памяти, забитый нулями, обозначает не ноль, а $0{,}1_2 = 0{,}5$. Таким образом, для нуля приходится использовать специальный способ хранения, используя для этого запись какого-нибудь другого числа.

Нужно понимать, что хотя в виде чисел с плавающей запятой представляются как громадные, так и близкие к нулю дробные числа, всё же в ограниченном участке памяти можно хранить лишь конечное количество комбинаций из нулей и единиц, то есть различных чисел с плавающей запятой не так уж и много. Если рассмотреть, как они расположены на числовой оси, то окажется, что большая часть этих чисел расположены около нуля, и чем дальше от нуля, тем реже начинают встречаться числа с плавающей запятой на числовой оси.

Конкретные способы хранения чисел с плавающей запятой в памяти компьютера и требования к вычислительным устройствам, поддерживающим работу с такими числами, описаны в стандарте IEEE 754. Этот стандарт дважды менялся. Последние изменения были внесены в 2008 году.

Три наиболее распространённых типа данных для хранения чисел с плавающей запятой приведены в таблице 8. Для каждого типа данных приведён его размер в битах и в байтах, название соответствующего типа данных в языке C++, а также примерное количество десятичных знаков точности, которое обеспечивает этот тип данных.

Таблица 8. Типы данных для хранения чисел с плавающей запятой

Бит	Байт	Название в C++	Десятичных знаков точности
32	4	`float`	7
64	8	`double`	15
80	10	`long double`	19

В действительности в стандарте языка C++ не указан явно размер этих типов данных, так же как и в случае с целочисленными типами данных, но на практике в большинстве реализаций компиляторов значения этих типов занимают именно столько памяти. Кроме того, тип данных `long double` может не поддерживаться на некоторых вычислительных устройствах. На практике в подавляющем большинстве случаев следует использовать тип данных `double`, представляющий собой число с плавающей запятой двойной точности.

2.2 Операции над числами с плавающей запятой

Основные арифметические операции с числами с плавающей запятой во многом совпадают с соответствующими операциями над целыми числами. Четыре основные бинарные операции: сложение (+), вычитание (–), умножение (*) и деление (/). Следует обратить внимание на операцию деления. В большинстве си-подобных языков программирования операция деления (/) обозначает одновременно и целочисленное деление, и деление с плавающей запятой, хотя это разные процессорные операции. Компилятор определяет, какую из них использовать, исходя из типа операндов: если оба операнда целые, то используется целочисленное деление, в противном случае используется деление с плавающей запятой.

Кроме стандартных арифметических операций большинство вычислительных устройств поддерживают вычисление многих элементарных функций для аргументов с плавающей запятой. В языках C и C++ многие такие функции объявлены в стандартном заголовочном файле `math.h`. Некоторые наиболее важные из них приведены в табл. 9.

Таблица 9. Основные функции для обработки чисел с плавающей запятой

Функция	**Описание**
`cos(x)`	Косинус
`sin(x)`	Синус
`tan(x)`	Тангенс
`acos(x)`	Арккосинус
`asin(x)`	Арксинус
`atan(x)`	Арктангенс
`atan2(y, x)`	Вычисление угла по полярным координатам
`cosh(x)`	Гиперболический косинус
`sinh(x)`	Гиперболический синус
`tanh(x)`	Гиперболический тангенс
`exp(x)`	Экспонента
`log(x)`	Натуральный логарифм
`pow(x, y)`	Возведение в степень
`sqrt(x)`	Квадратный корень
`ceil(x)`	Округление вверх до ближайшего целого
`floor(x)`	Округление вниз до ближайшего целого
`fabs(x)`	Модуль

Нужно понимать, что эти функции принимают в качестве аргументов числа с плавающей запятой и возвращают также числа с плавающей запятой. При передаче целых чисел в эти функции они будут приведены к типу `double`. При этом точность может потеряться. Таким образом, например, не следует использовать функцию `pow` для возведения целых чисел в степень целых чисел.

Главное, что следует понимать при работе с числами с плавающей запятой: каждая операция с такими числами, какая бы она ни была, может привести к небольшой вычислительной ошибке. Если точный результат, который должен получиться в результате операции, не представляется в виде числа с плавающей запятой, то результатом станет некоторое близкое к нему число с плавающей запятой. Таким образом, последующие операции примут на вход уже не совсем точный результат предыдущих операций, в результате чего ошибка может ещё больше увеличиться. Это означает, что на практике следует стараться решать задачу в целых числах, если это возможно.

Также можно заметить ещё одну важную особенность, вытекающую из неточности выполнения операций. Поскольку операция сравнения двух чисел на равенство «==» проверяет точное равенство, два числа с плавающей запятой, полученные в результате различных операций, могут немного отличаться из-за погрешности вычислений, даже если они в точности равны с точки зрения математики.

Отсюда следует важное правило: никогда нельзя сравнивать два числа с плавающей запятой на точное равенство. Вместо этого следует вычитать их друг из друга и сравнивать модуль разности с некоторым достаточно маленьким числом. Пример такого подхода к сравнению приведён в листинге 6.

Листинг 6. Сравнение чисел с плавающей запятой

```
const double EPS = 1.0E-6;

bool is_equal(double x, double y)
{
    return abs(x - y) < EPS;
}
```

Выбор самого числа EPS является отдельной задачей. С одной стороны оно должно быть достаточно мало, чтобы разные числа ошибочно не полагались одинаковыми, но с другой стороны оно должно быть больше возможной погрешности вычислений с плавающей запятой. Также следует понимать, что такой способ сравнения всё равно не обеспечивает никакой гарантии, что сравнение пройдёт правильно. Он всё ещё допускает ошибки в обе стороны. Вот почему следует с особенной осторожностью работать с числами с плавающей запятой.

2.3 Бесконечность и не число

Кроме положительного и отрицательного нулей существует еще, по меньшей мере, три специальных значения, которые могут храниться вместо соответствующих чисел с плавающей запятой: положительная бесконечность, отрицательная бесконечность и не число. Эти значения представляют собой предельные продолжения множества вещественных чисел, связанные с известным в математике понятием расширенной числовой прямой. Наличие этих значений позволяет построить арифметику с плавающей запятой, практически не приводящую к падению программ с ошибками из-за некорректных операций.

Бесконечность может возникнуть в программе в результате переполнения с плавающей запятой. Эта ситуация возникает когда значение порядка в результате операции претерпевает обычное целочисленное переполнение. Если значение порядка переполняется в отрицательную сторону, то получается ноль: положительный, если результат должен быть положительным, и отрицательный, если результат должен быть отрицательным. Если значение порядка переполняется в положительную сторону, то получается бесконечность: поло-

жительная, если результат должен быть положительным, и отрицательная, если результат должен быть отрицательным.

Бесконечности корректно ведут себя с большинством арифметических операций, они вступают с другими числами с плавающей запятой в естественное отношение порядка. Однако при сложении любого конечно числа с бесконечностью она не меняется, что может привести к потере данных. Кроме того, ниже указано, что работа с бесконечностями может приводить к возникновению не чисел.

Не число может возникнуть по трём основным причинам:

1) выполнение арифметической операции, одним из операндов которой является не число,

2) выполнение арифметической операции, аналог предела для которой представляет собой неопределённость,

3) вычисление элементарной функции, приводящее к мнимым результатам.

Не число приводит к возникновению ещё большего количества не чисел в результате любых арифметических операций. Кроме того, все операции сравнения с не числами возвращают ложь. Это приводит к большим проблемам с корректной обработкой этих значений. Даже чтобы проверить, хранится ли в некоторой переменной не число, нужно сравнивать эту переменную с самой собой на точное равенство (только не число не равно самому себе).

Таким образом, не рекомендуется намеренно использовать бесконечности и не числа в своих программах. Напротив, в подавляющем большинстве случаев следует всячески избегать возникновения этих специальных значений, а их появление обычно означает наличие ошибки в исходном коде программы. Потенциальное наличие в программе подобных ошибок следует иметь в виду в процессе тестирования и отладки программы.

2.4 Задание на лабораторную работу № 2

Вариант 0. A+B

Программа должна складывать два дробных числа.

Входные данные. Два дробных числа.

Выходные данные. Сумма этих чисел.

Пример текстового интерфейса пользователя

```
A+B
Input a decimal>1.2
Input a decimal>3.4
Result: 4.6
Continue? (Y/N)>Y
Input a decimal>2
Input a decimal>2
Result: 4.0
Continue? (Y/N)>N
```

Вариант 1. Из десятичной системы в двоичную

Программа должна переводить дробное число из десятичной системы в двоичную.

Входные данные. Дробное число в десятичной системе счисления.

Выходные данные. То же дробное число в двоичной системе счисления.

Пример текстового интерфейса пользователя

```
Translating a number from decimal to binary
Input a decimal>5.625
Result: 101.101
Continue? (Y/N)>Y
Input a decimal>0.1
Result: 0.0001100110011001100110011001100110011001100110011...
Continue? (Y/N)>N
```

Вариант 2. Из двоичной системы в десятичную

Программа должна переводить дробное число из двоичной системы в десятичную.

Входные данные. Дробное число в двоичной системе счисления.

Выходные данные. То же дробное число в десятичной системе счисления.

Пример текстового интерфейса пользователя

```
Translating a number from binary to decimal
Input a decimal>101.101
Result: 5.625
Continue? (Y/N)>Y
Input a decim-
al>0.0001100110011001100110011001100110011
Result: 0.09999999999854481
Continue? (Y/N)>N
```

Вариант 3. Площадь треугольника

Программа должна вычислять площадь треугольника, заданного целочисленными координатами своих вершин.

Входные данные. Целочисленные координаты трёх вершин треугольника.

Выходные данные. Единственное дробное число – площадь треугольника.

Пример текстового интерфейса пользователя

```
Area of a triangle
Input a vertex>-1 2
Input a vertex>1 3
Input a vertex>4 -2
Result: 6.5
Continue? (Y/N)>Y
Input a vertex>-1 -1
Input a vertex>0 2
Input a vertex>1 -1
Result: 3.0
Continue? (Y/N)>N
```

Вариант 4. Геометрическая прогрессия

Программа должна вычислять сумму заданного количества начальных элементов заданной геометрической прогрессии.

Входные данные. Первый элемент геометрической прогрессии, знаменатель этой прогрессии и количество элементов.

Выходные данные. Единственное дробное число – сумма заданного количества элементов этой геометрической прогрессии.

Пример текстового интерфейса пользователя

```
Geometric series
Input a start value>2
Input a common ratio>3
Input the number of elements>4
Result: 80.0
Continue? (Y/N)>Y
Input a start value>0.1
Input a common ratio>0.5
Input the number of elements>1000000
Result: 0.2
Continue? (Y/N)>N
```

Вариант 5. Приближённый корень

Программа должна для заданного неотрицательного числа x вычислять элементы последовательности, первый элемент которой равен x, а каждый последующий – среднему арифметическому из предыдущего элемента и отношения x к предыдущему элементу, до тех пор, пока очередной элемент не станет отличаться от квадратного корня из x не более чем на заданное число.

Входные данные. Неотрицательное дробное число x, из которого нужно извлечь корень, и положительное дробное значение точности вычислений.

Выходные данные. Номер первого подходящего элемента последовательности, сам этот элемент и значение модуля разности между этим элементом и реальным значением корня из x.

Пример текстового интерфейса пользователя

```
Computing square root
Input a nonnegative decimal>4
Input an upper bound of the absolute error>0.1
Result: 2.05
Element #3
Absolute error: 0.05
Continue? (Y/N)>Y
Input a nonnegative decimal>1.44
Input an upper bound of the absolute error>0.0001
Result: 1.200000011196177
Element #4
Absolute error: 1.1196177096550741e-08
Continue? (Y/N)>N
```

Вариант 6. Система линейных алгебраических уравнений

Программа должна решать систему линейных алгебраических уравнений вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1; \\ a_{21}x_1 + a_{22}x_2 = b_2. \end{cases}$$

Входные данные. Целые числа a_{ij} и b_i.

Выходные данные. Все возможные пары значений x_1 и x_2, удовлетворяющие обоим уравнениям системы, либо сообщение о том, что таких пар бесконечно много.

Пример текстового интерфейса пользователя

```
System of linear equations solver
Input a(1, 1)>1
Input a(1, 2)>2
Input a(2, 1)>3
Input a(2, 2)>4
Input b(1)>5
Input b(2)>6
```

```
Solution: (-4.0, 4.5)
Continue? (Y/N)>Y
Input a(1, 1)>1
Input a(1, 2)>2
Input a(2, 1)>2
Input a(2, 2)>4
Input b(1)>3
Input b(2)>3
No solution
Continue? (Y/N)>N
```

Вариант 7. Перевод в полярную систему координат

Программа должна переводить декартовы координаты точки на плоскости в полярную систему координат, представляющую собой расстояние от начала координат до этой точки и угол, который образует радиус-вектор точки с положительным направлением оси абсцисс.

Входные данные. Координаты точки в декартовой системе координат.

Выходные данные. Расстояние от начала координат до этой точки и угол в радианах, который образует радиус-вектор этой точки с положительным направлением оси абсцисс.

Пример текстового интерфейса пользователя

```
Translation from Cartesian to polar coordinate system
Input x>1
Input y>2
Distance: 2.23606797749979
Angle: 1.1071487177940904
Continue? (Y/N)>Y
Input x>0.3
Input y>-0.4
Distance: 0.5
Angle: 5.355890089177974
Continue? (Y/N)>N
```

Вариант 8. Поиск пары ближайших точек

Программа должна находить пару ближайших точек в заданном конечном наборе точек на плоскости.

Входные данные. Конечный набор точек на плоскости, заданных своими декартовыми координатами.

Выходные данные. Номера двух точек, евклидово расстояние между которыми минимально, а также само это расстояние.

Пример текстового интерфейса пользователя

```
Nearest points
Input the number of points>3
Input coordinates of the point #1>0 0
Input coordinates of the point #2>2 1
Input coordinates of the point #3>1 2
Nearest points: 2 and 3
Distance: 1.4142135623730951
Continue? (Y/N)>Y
Input the number of points>4
Input coordinates of the point #1>1.5 1.5
Input coordinates of the point #2>-0.5 5.5
Input coordinates of the point #3>3.5 3.5
Input coordinates of the point #4>5.5 -0.5
Nearest points: 1 and 3
Distance: 2.8284271247461903
Continue? (Y/N)>N
```

Вариант 9. Поиск пары наиболее удалённых точек

Программа должна находить пару наиболее удалённых точек в заданном конечном наборе точек на плоскости.

Входные данные. Конечный набор точек на плоскости, заданных своими декартовыми координатами.

Выходные данные. Номера двух точек, евклидово расстояние между которыми максимально, а также само это расстояние.

Пример текстового интерфейса пользователя

```
Most distant points
Input the number of points>3
Input coordinates of the point #1>0 0
Input coordinates of the point #2>4 1
Input coordinates of the point #3>1 4
Nearest points: 2 and 3
Distance: 4.242640687119286
Continue? (Y/N)>Y
Input the number of points>4
Input coordinates of the point #1>1.5 1.5
Input coordinates of the point #2>-0.5 5.5
Input coordinates of the point #3>3.5 3.5
Input coordinates of the point #4>5.5 -0.5
Nearest points: 2 and 4
Distance: 8.485281374238571
Continue? (Y/N)>N
```

Вариант 10. Принадлежность точки треугольнику

Программа должна проверять, лежит ли заданная точка внутри заданного треугольника.

Входные данные. Декартовы координаты точки на плоскости и декартовы координаты вершин треугольника.

Выходные данные. Сообщение о том, лежит ли заданная точка внутри треугольника, или же она находит за его пределами.

Пример текстового интерфейса пользователя

```
Check if a point is inside a triangle
Input coordinates of the vertex #1>1 1
Input coordinates of the vertex #2>4 1
Input coordinates of the vertex #3>1 4
Input coordinates of the point>2 1
The point is inside the triangle
Continue? (Y/N)>Y
```

```
Input coordinates of the vertex #1>1.5 1.5
Input coordinates of the vertex #2>4.5 1.5
Input coordinates of the vertex #3>1.5 4.5
Input coordinates of the point>3.5 2.5
The point is outside the triangle
Continue? (Y/N)>N
```

Вариант 11. Построение треугольника

Программа должна строить на плоскости треугольник с заданными длинами сторон.

Входные данные. Три дробных положительных числа – длины сторон треугольника.

Выходные данные. Координаты трёх точек на плоскости, являющихся вершинами треугольника с такими длинами сторон.

Пример текстового интерфейса пользователя

```
Triangle construction
Input a positive decimal>3
Input a positive decimal>4
Input a positive decimal>5
Triangle vertices:
(0.0, 0.0)
(3.0, 0.0)
(0.0, 4.0)
Continue? (Y/N)>Y
Input a positive decimal>2.23606797749979
Input a positive decimal>2.23606797749979
Input a positive decimal>1.4142135623730951
Triangle vertices:
(1.0, 1.0)
(3.0, 2.0)
(2.0, 3.0)
Continue? (Y/N)>N
```

Вариант 12. Построение прямоугольника

Программа должна строить четвёртую вершину прямоугольника по трём другим вершинам этого прямоугольника.

Входные данные. Декартовы координаты трёх вершин прямоугольника на плоскости.

Выходные данные. Координаты четвёртой точки этого прямоугольника.

Пример текстового интерфейса пользователя

```
Rectangle construction
Input coordinates of the vertex #1>0 0
Input coordinates of the vertex #2>3 0
Input coordinates of the vertex #3>0 4
Vertex #4: (3.0, 4.0)
Continue? (Y/N)>Y
Input coordinates of the vertex #1>1.5 2.5
Input coordinates of the vertex #2>3.5 5.5
Input coordinates of the vertex #3>9.5 1.5
Vertex #4: (7.5, -1.5)
Continue? (Y/N)>N
```

Вариант 13. Расстояние от точки до окружности

Программа должна определять расстояние от данной точки до окружности, заданной координатами центра и радиусом. Расстояние от точки до окружности – это наименьшее из расстояний от этой точки до точек этой окружности.

Входные данные. Декартовы координаты точки на плоскости, декартовы координаты центра окружности на этой же плоскости и положительный радиус этой окружности.

Выходные данные. Расстояние от точки до окружности.

Пример текстового интерфейса пользователя

```
Distance between a point and a circle
Input coordinates of a point>4 6
Input coordinates of a centre>1 2
Input a radius>3
Distance: 2.0
Continue? (Y/N)>Y
Input coordinates of a point>3.5 1.5
Input coordinates of a centre>1.5 2.5
Input a radius>3.5
Distance: 1.2639320225002102
Continue? (Y/N)>N
```

Вариант 14. Вписанная окружность

Программа должна строить окружность, вписанную в треугольник, заданный своими вершинами.

Входные данные. Декартовы координаты вершин треугольника на плоскости.

Выходные данные. Координаты центра вписанной окружности и её радиус.

Пример текстового интерфейса пользователя

```
Incircle of a triangle
Input coordinates of the vertex #1>1 1
Input coordinates of the vertex #2>3 3
Input coordinates of the vertex #3>-1 5
Centre: (1.2792407799438736, 2.7207592200561264)
Radius: 1.019307464208828
Continue? (Y/N)>Y
Input coordinates of the vertex #1>2.5 1.5
Input coordinates of the vertex #2>3.5 6.5
Input coordinates of the vertex #3>4.5 -1.5
Centre: (3.323267815490337, 1.6628631097680628)
Radius: 0.775340427143804
Continue? (Y/N)>N
```

Вариант 15. Описанная окружность

Программа должна строить окружность, описанную около треугольника, заданного своими вершинами.

Входные данные. Декартовы координаты вершин треугольника на плоскости.

Выходные данные. Координаты центра описанной окружности и её радиус.

Пример текстового интерфейса пользователя

```
Circumscribed circle
Input coordinates of the vertex #1>1 1
Input coordinates of the vertex #2>3 3
Input coordinates of the vertex #3>-1 5
Circumcenter: (0.6666666666666666, 3.3333333333333335)
Radius: 2.3570226039551585
Continue? (Y/N)>Y
Input coordinates of the vertex #1>2.5 1.5
Input coordinates of the vertex #2>3.5 6.5
Input coordinates of the vertex #3>4.5 -1.5
Circumcenter: (8.0, 3.0)
Radius: 5.70087712549569
Continue? (Y/N)>N
```

Вариант 16. Площадь пересечения двух прямоугольников

Программа должна вычислять площадь пересечения двух прямоугольников на плоскости, заданных координатами верхнего левого и нижнего правого углов, стороны которых параллельны осям координат.

Входные данные. Координаты левых верхних и нижних правых углов двух прямоугольников, стороны которых параллельны осям координат.

Выходные данные. Площадь пересечения внутренних частей этих прямоугольников.

Пример текстового интерфейса пользователя

```
Rectangles intersection area
Input coordinates of the upper left vertex #1>-2 4
Input coordinates of the bottom right vertex #1>3 2
Input coordinates of the upper left vertex #2>1 3
Input coordinates of the bottom right vertex #2>4 0
Intersection area: 2.0
Continue? (Y/N)>Y
Input coordinates of the upper left vertex #1>-2.0 1.0
Input coordinates of the bottom right vertex #1>4 -1
Input coordinates of the upper left vertex #2>1.2 3.4
Input coordinates of the bottom right vertex #2>2 -2
Intersection area: 1.6
Continue? (Y/N)>N
```

Вариант 17. Подобие треугольников

Два треугольника называются подобными, если один из них можно получить из другого преобразованиями сдвига, поворота и масштабирования. Программа должна определять, являются ли подобными два треугольника на плоскости, заданные координатами своих вершин.

Входные данные. Декартовы координаты вершин двух треугольников на плоскости.

Выходные данные. Сообщение о том, являются ли эти два треугольника подобными или не являются.

Пример текстового интерфейса пользователя

```
Similar triangles check
Input the first triangle
Input coordinates of the vertex #1>1 1
Input coordinates of the vertex #2>3 3
Input coordinates of the vertex #3>-1 5
Input the second triangle
Input coordinates of the vertex #1>-1 -1
Input coordinates of the vertex #2>1 3
```

```
Input coordinates of the vertex #3>3 1
Triangles are similar
Continue? (Y/N)>Y
Input the first triangle
Input coordinates of the vertex #1>1.5 1.5
Input coordinates of the vertex #2>3.5 3.5
Input coordinates of the vertex #3>-1.5 5.5
Input the second triangle
Input coordinates of the vertex #1>0.5 0.5
Input coordinates of the vertex #2>1.5 0.5
Input coordinates of the vertex #3>0.5 1.5
Triangles are not similar
Continue? (Y/N)>N
```

Вариант 18. Построение параболы

Программа должна строить уравнение параболы по трём заданным точкам на плоскости.

Входные данные. Декартовы координаты трёх точек на плоскости.

Выходные данные. Три коэффициента в уравнении параболы.

Пример текстового интерфейса пользователя

```
Parabola construction
Input coordinates of the point #1>1 1
Input coordinates of the point #2>2 2
Input coordinates of the point #3>3 5
y = 1.0 x^2 - 2.0 x + 2.0
Continue? (Y/N)>Y
Input coordinates of the point #1>1.5 0.5
Input coordinates of the point #2>3.5 0.5
Input coordinates of the point #3>4.5 -2.5
y = -1.0 x^2 + 5.0 x - 4.75
Continue? (Y/N)>N
```

Вариант 19. Кубическое уравнение

Программа должна находить все вещественные корни кубического уравнения $x^3 + a\,x^2 + b\,x + c = 0$, заданного своими коэффициентами a, b и c.

Входные данные. Три дробных коэффициента в кубическом уравнении.

Выходные данные. Все вещественные корни кубического уравнения.

Пример текстового интерфейса пользователя

```
Cubic equation solution
Input a coefficient at x^2>-6
Input a coefficient at x>11
Input a constant term>-6
Roots: 1.0, 2.0, 3.0
Continue? (Y/N)>Y
Input a coefficient at x^2>0.5
Input a coefficient at x>0.5
Input a constant term>-0.5
Roots: 0.5
Continue? (Y/N)>N
```

Вариант 20. Процент по вкладу

Программа должна вычислять сумму денег, которая будет лежать в банке через заданное количество месяцев, исходя из начальной суммы и процента по вкладу, начисляемого через каждый месяц.

Входные данные. Начальная сумма в рублях, ставка по вкладу в процентах и количество месяцев.

Выходные данные. Сумма в рублях, которая будет лежать в банке через заданное количество месяцев.

Пример текстового интерфейса пользователя

```
Deposit percentage
Input an initial deposit>100
Input a monthly deposit rate>0.05
Input the number of months>12
```

```
Result: 179.59
Continue? (Y/N)>Y
Input an initial deposit>500000.50
Input a monthly deposit rate>0.01
Input the number of months>6
Result: 530760.61
Continue? (Y/N)>N
```

Вариант 21. Концентрация раствора

Заданное количество раствора некоторого вещества в воде имеет заданную концентрацию. Программа должна выяснять, как много воды или вещества нужно добавить, чтобы получить другую заданную концентрацию.

Входные данные. Текущий объём раствора, текущая объёмная концентрация вещества в растворе и требуемая объёмная концентрация вещества в растворе.

Выходные данные. Объём воды либо вещества, который нужно добавить в раствор, чтобы получилась требуемая концентрация.

Пример текстового интерфейса пользователя

```
Concentration calculator
Input a solution volume>1000
Input an initial concentration>0.96
Input a required concentration>0.40
Add 1400.0 volume units of water
Continue? (Y/N)>Y
Input a solution volume>0.1
Input an initial concentration>0.40
Input a required concentration>0.96
Add 1.4 volume units of a solute
Continue? (Y/N)>N
```

Вариант 22. Два сплава

Имеется два сплава из двух металлов, соотношение этих металлов в каждом из сплавов известно. Программа должна определять, какое количество объёмных единиц первого и второго сплава нужно взять, чтобы получить третий сплав с заданным соотношением этих металлов.

Входные данные. Соотношение двух металлов в первом сплаве, во втором сплаве и в требуемом сплаве, а также количество объёмных единиц третьего сплава, которое требуется получить.

Выходные данные. Количество объёмных единиц первого и второго сплава, которые следует соединить, для получения заданного количества третьего сплава, либо сообщение о том, что получить третий сплав из имеющихся двух невозможно.

Пример текстового интерфейса пользователя

```
Alloys volume calculator
Input a proportion in the first alloy>0.25
Input a proportion in the second alloy>4.0
Input a required proportion>1.0
Input a required volume>1000
Volume of the first alloy: 500.0
Volume of the second alloy: 500.0
Continue? (Y/N)>Y
Input a proportion in the first alloy>0.5
Input a proportion in the second alloy>2.0
Input a required proportion>0.666666666666666
Input a required volume>1.5
Volume of the first alloy: 1.2
Volume of the second alloy: 0.3
Continue? (Y/N)>N
```

Вариант 23. Задача о поездах

Из пункта А в пункт Б выехал поезд, движущийся с заданной постоянной скоростью v_1 км/ч. Одновременно навстречу ему из пункта Б в пункт А выехал поезд, движущийся с заданной постоянной скоростью v_2 км/ч. Программа должна определять, через какое время и в

какой точке они встретятся, если расстояние между пунктами А и Б составляет s км.

Входные данные. Скорости поездов v_1 и v_2, а также расстояние s.

Выходные данные. Расстояние до точки встречи от пункта А и время в часах, через которое поезда одновременно окажутся в этой точке.

Пример текстового интерфейса пользователя

```
Train problem
Input the first velocity>60
Input the second velocity>90
Input a distance between A and B>100
Distance from A to the meeting point: 40.0
Time to meeting: 0.666666666666667
Continue? (Y/N)>Y
Input the first velocity>100.0
Input the second velocity>50.0
Input a distance between A and B>400.0
Distance from A to the meeting point:
266.6666666666667
Time to meeting: 2.666666666666667
Continue? (Y/N)>N
```

Вариант 24. Задача о велосипедах

Из пункта А выехал велосипедист, движущийся с заданной постоянной скоростью v_1 км/ч. Через t часов после этого вслед за ним из пункта А выехал велосипедист, движущийся с заданной постоянной скоростью v_2 км/ч. Программа должна определять, через какое время и в какой точке второй велосипедист догонит первого.

Входные данные. Скорости велосипедистов v_1 и v_2, а также время t.

Выходные данные. Расстояние до точки встречи от пункта А и время в часах, через которое велосипедисты одновременно окажутся в этой точке.

Пример текстового интерфейса пользователя

```
Cycle problem
Input the first velocity>20
Input the second velocity>30
Input delay time>10
Distance from A to the meeting point: 600.0
Time to meeting: 20.0
Continue? (Y/N)>Y
Input the first velocity>10.5
Input the second velocity>15.5
Input delay time>0.5
Distance from A to the meeting point: 16.275
Time to meeting: 1.05
Continue? (Y/N)>N
```

Вариант 25. Задача о землекопах

Один землекоп выкапывает *s*-метровую траншею за *t* часов. Программа должна определять, сколько времени понадобится *n* землекопам, чтобы выкопать *l*-метровую траншею.

Входные данные. Положительное число *s*, положительное число *t*, целое положительное число *n*, положительное число *l*.

Выходные данные. Время в часах, за которое *n* землекопов смогут выкопать *l*-метровую траншею.

Пример текстового интерфейса пользователя

```
Digger problem
Input trench length s>10
Input time t for one digger>20
Input the number of diggers n>5
Input target trench length l>40
Time required: 16.0
Continue? (Y/N)>Y
Input trench length s>0.5
Input time t for one digger>1.5
Input the number of diggers n>10
Input target trench length l>2.5
Time required: 0.75
Continue? (Y/N)>N
```

Вариант 26. Задача о бассейне

Имеется несколько труб, для каждой из которых известно, за сколько часов данная труба полностью заполняет бассейн водой. Программа должна определять, за сколько часов бассейн заполнится, если задействовать все трубы одновременно.

Входные данные. Конечный набор положительных чисел, обозначающих время в часах, за которое очередная труба полностью заполняет водой бассейн.

Выходные данные. Время в часах, за которое бассейн заполнится водой, если задействовать все трубы одновременно.

Пример текстового интерфейса пользователя

```
Swimming pool problem
Input the number of pipes>3
Input filling time for the pipe #1>10
Input filling time for the pipe #2>20
Input filling time for the pipe #3>30
Time required: 5.454545454545454
Continue? (Y/N)>Y
Input the number of pipes>4
Input filling time for the pipe #1>0.25
Input filling time for the pipe #2>0.50
Input filling time for the pipe #3>0.75
Input filling time for the pipe #4>1.00
Time required: 0.12
Continue? (Y/N)>N
```

Вариант 27. Последняя секунда падения

За последнюю секунду падения тело прошло заданную долю своего пути. Программа должна определять, с какой высоты падало это тело. Можно считать, что падение происходило на Земле, так что ускорение свободного падения составляет 9,8 м/с^2.

Входные данные. Дробное число из интервала (0; 1) – доля расстояния, которую падающее тело прошло за последнюю секунду падения.

Выходные данные. Высота в метрах, с которой падало тело.

Пример текстового интерфейса пользователя

```
Last second of the fall
Input a fraction of distance covered in the last
second>0.5
Total height: 57.118585822512664
Continue? (Y/N)>Y
Input a fraction of distance covered in the last
second>0.1
Total height: 1860.7096320895032
Continue? (Y/N)>N
```

Вариант 28. Винни-Пух и все-все-все

Пятачок как-то поссорился с Винни-Пухом и решил каждый день с утра добавлять ему в горшочек мёда ложку дёгтя. Вечером каждого дня Винни-Пух съедал половину содержимого горшочка. Программа должна определять, сколько ложек дёгтя съел Винни-Пух за заданное количество дней. Следует считать, что дёготь полностью растворяется в мёде, образуя однородный раствор.

Входные данные. Целое положительное количество дней, которое Винни-Пух ел мёд.

Выходные данные. Общее количество ложек дёгтя, которое Винни-Пух съел за это время.

Пример текстового интерфейса пользователя

```
Winnie-the-Pooh problem
Input the number of days>2
Total number of tar spoons: 1.25
Continue? (Y/N)>Y
Input the number of days>3
Total number of tar spoons: 2.125
Continue? (Y/N)>N
```

Вариант 29. Материальная точка

Программа должна моделировать поведение материальной точки заданной массы на прямой. Пользователь может периодически воздействовать на точку заданной силой в заданном направлении заданное время, а программа должна выводить координаты точки по истечении этого времени. Между взаимодействиями точка должна сохранять свою массу, текущую координату и скорость.

Входные данные. Масса точки и набор воздействий, каждое из которых включает силу, направление и время воздействия.

Выходные данные. После каждого воздействия программа должна выводить текущую координату точки.

Пример текстового интерфейса пользователя

```
Linear motion model
Input a mass of a body>5
Input force (positive or negative)>2
Input time of influence>20
Current position: 80.0
Continue? (Y/N)>Y
Input force (positive or negative)>-3.2
Input time of influence>5.5
Current position: 114.32
Continue? (Y/N)>N
```

Вариант 30. Калькулятор дробных чисел

Состоянием программы является дробное число. Пользователь может выполнять с этим числом арифметические операции: сложение, вычитание, умножение и деление, изменяя состояние. Программа должна визуализировать результат каждой операции.

Входные данные. Операции и дробные операнды.

Выходные данные. Результат каждой операции.

Пример текстового интерфейса пользователя

```
Decimal calculator
Initial state: 0.0
Input an operation (+/-/*//)>+
Input an operand>7.1
Result: 7.1
Continue? (Y/N)>Y
Input an operation (+/-/*//)>/
Input an operand>3.2
Result: 2.2187499999999996
Continue? (Y/N)>N
```

2.5 Контрольные вопросы

1. Как числа с плавающей запятой хранятся в памяти компьютера?
2. Какие типы данных для хранения чисел с плавающей запятой поддерживаются вычислительными устройствами?
3. Сколько десятичных знаков точности обеспечивается 64-битным типом данных для хранения чисел с плавающей запятой двойной точности?
4. Какие операции можно выполнять над числами с плавающей запятой, и какие функции для них определены? Чем они отличаются от математических аналогов?
5. Каким образом накапливается погрешность при вычислениях с плавающей запятой?
6. Как правильно сравнивать два числа с плавающей запятой на точное равенство?
7. Что такое переполнение с плавающей запятой, в каком случае оно возникает, и что получается в результате?
8. Каким образом при обработке чисел с плавающей запятой возникают положительные и отрицательные бесконечности, каковы их свойства?
9. Каким образом при обработке чисел с плавающей запятой возникают не числа, каковы их свойства?
10. Как проверить, хранит ли переменная не число?

2.6 Пример выполнения лабораторной работы

В листинге 7 приведён пример исходного кода выполненного нулевого варианта настоящей лабораторной работы. Как видно, большая часть кода опять же посвящена общению с пользователем и контролю ввода. Ещё раз следует заметить, что для успешной сдачи лабораторной работы нет необходимости реализовывать интерфейс пользователя и защиту от ввода: достаточно просто заменить `int` на `double` в листинге 2, и этого будет достаточно для успешной сдачи нулевого варианта второй лабораторной работы. Однако в этом случае общая оценка за работу будет ниже.

Листинг 7. Пример исходного кода выполненной лабораторной работы

```
#include <iostream>
#include <string>

const char EOLN = '\n';
const char YES_CHAR = 'Y';
const char NO_CHAR = 'N';
const std::string ABOUT_MESSAGE = "A+B";
const std::string CONTINUE_MESSAGE = "Continue? (Y/N)>";
const std::string INCORRECT_MESSAGE =
"Input is incorrect. Try again>";
const std::string INPUT_MESSAGE = "Input a decimal>";
const std::string OUTPUT_MESSAGE = "Result: ";
const std::string SKIP_CHARACTERS = " ";

void ClearInputStream(std::istream &in)
{
    in.clear();
    while (in.peek() != EOLN && in.peek() != EOF)
    {
        in.get();
    }
}

int Seek(std::istream &in)
{
    while (in.peek() != EOLN &&
SKIP_CHARACTERS.find((char)in.peek()) != std::string::npos)
    {
        in.get();
    }
```

```
    return in.peek();
}

double ReadFloat(std::istream &in)
{
    std::cout << INPUT_MESSAGE;
    double ans;
    in >> ans;
    while (!in || Seek(in) != EOLN)
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    return ans;
}

bool NeedContinue(std::istream &in)
{
    std::cout << CONTINUE_MESSAGE;
    char ans;
    in >> ans;
    while (!in || Seek(in) != EOLN || ans != YES_CHAR && ans
!= NO_CHAR)
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    return ans == YES_CHAR;
}

int main()
{
    std::cout << ABOUT_MESSAGE << std::endl;
    bool cont = true;
    while (cont)
    {
        double a = ReadFloat(std::cin);
        double b = ReadFloat(std::cin);
        std::cout << OUTPUT_MESSAGE << a + b << std::endl;
        cont = NeedContinue(std::cin);
    }
    return 0;
}
```

3 Работа с массивами

3.1 Понятие массива

Массив – это набор занумерованных элементов одного типа, расположенных в памяти непосредственно друг за другом. Оперативная память допускает произвольный доступ, что означает, что можно быстро узнать содержимого любого байта, зная его адрес, то есть номер. Таким образом, зная адрес первого элемента массива и размер каждого элемента, можно вычислить адрес любого другого элемента по его номеру.

Переменная, хранящая массив, фактически хранит адрес первого байта первого элемента массива в памяти. Массив как тип данных определяет вид массива и тип его элементов. Массив как структура данных определяет, что эти элементы размещаются в памяти непосредственно друг за другом в том порядке, в котором они занумерованы. Поскольку эти элементы однотипные, все они имеют одинаковый размер, и за счёт этого можно организовать быстрый доступ к каждому отдельному элементу по его номеру.

Основная особенность массива, как уже упоминалось, – это возможность быстро узнать значение элемента, зная его номер. Это связано с тем, что для этого не нужно искать положение этого элемента в памяти, а вместо этого можно просто вычислить его адрес. Кроме того, в силу расположения элементов последовательный доступ к ним в порядке их нумерации также чуть быстрее, чем последовательный доступ к такому же количеству элементов, физически расположенных в памяти в произвольных местах.

Массив предназначен для хранения набора из нескольких однотипных значений, особенно когда количество этих значений неизвестно до запуска программы. Таким образом, вместо хранения не-

скольких переменных вида x_1, x_2, x_3 и т.д., можно завести один массив x и точно так же получать доступ к отдельным значениям x_k по их номеру. Более того, номер элемента сам по себе теперь можно хранить в отдельной переменной, чтобы единообразно обрабатывать все элементы в одном цикле.

На рис. 5 показан пример хранения массива из трёх 32-битных целых чисел в памяти компьютера. Его первый элемент хранится в ячейках памяти с адресами с 14 по 17 включительно, так что адресом массива, хранящимся в переменной `a`, будет число 14. Сверху написаны адреса соответствующих ячеек памяти. Видно, что второй элемент занимает байты с адресами с 18 по 21, а третий – с 22 по 25. Следует отметить, что в языках программирования обычно элементы массивов нумеруются с нуля.

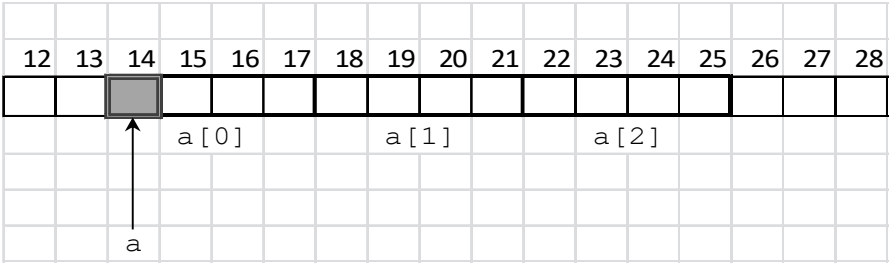

Рис. 5. Пример хранения массива в оперативной памяти

Иногда массивы называют векторами из-за сходства с соответствующим математическим объектом. На самом деле, это сходство невелико, поскольку математические операции над векторами, как правило, не определены над массивами. Правильнее было бы рассматривать массив просто как набор элементов, аналогом которого в математике выступает кортеж. Тем не менее, в некоторых языках программирования кортежи – это отдельные высокоуровневые структуры данных со специальными синтаксическими особенностями, которые хотя и основаны на массивах, работают иначе.

Массив – это фундаментальная структура данных, достаточно низкоуровневая, чтобы её преимущества проявлялись на подавляющем большинстве вычислительных устройств. Это сделало её очень популярной, так что в большинстве языков программирования есть возможность работы с массивами на уровне синтаксиса. Даже если в языке программирования очень высокого уровня отсутствует структура данных массив, как таковая, аналогичные линейные структуры данных, основанные по своей природе на массивах, всё равно, как правило, присутствуют.

Также массив называют фундаментальной структурой данных, потому что многие структуры данных более высокого уровня основаны на массивах и строятся с их помощью. Массивы активно используются для хранения хэш-таблиц, а также графов в различных формах. Многие алгоритмы на графах предполагают, что информация о графе полностью или частично хранится в массивах.

3.2 Реализация массивов в языке C++

В C++, как и во многих других языках, существует разделение массивов на статические и динамические в зависимости от области памяти, в которой хранятся эти массивы. *Статические массивы* хранятся в статической памяти (на стеке) и отличаются постоянным размером, который должен быть известен на момент компиляции программы. *Динамические массивы* хранятся в динамической памяти (в куче) и отличаются возможностью задать необходимый размер прямо в ходе работы программы.

Статическую память можно понимать, как сегмент оперативной памяти, выделенной программе в начале её работы, в котором переменные хранятся на всём протяжении работы программы. При вызове функции в стек добавляется заданный объём памяти, необходимый для хранения переменных, а при выходе из функции он освобождается. Все переменные по умолчанию создаются на стеке. Работа

со стеком несколько быстрее, чем с кучей, за счёт особенностей реализации этого сегмента памяти, но существует жёсткое ограничение, обязывающее программу всегда заранее понимать, сколько статической памяти она использует на каждом этапе своей работы и что именно она в ней хранит. Из-за этого размер статических массивов должен быть точно задан на этапе компиляции программы.

В C++, как и в C, статические массивы объявляются с помощью конструкции

тип идентификатор[размер],

которая создаёт переменную идентификатор, хранящую массив из размер элементов типа тип. В качестве размера может выступать литерал, константа либо константное выражение, которое может быть вычислено компилятором в момент компиляции. Таким образом, нельзя считать переменную с клавиатуры и создать статический массив такого размера. Использовать статические массивы имеет смысл, если заранее известно точное количество элементов.

Листинг 8. Пример работы со статическим массивом в языке C++

```
#include <iostream>

const int N = 10;

int main()
{
	int a[N];
	for (int i = 0; i < N; ++i)
	{
		std::cin >> a[i];
	}
	for (int i = 0; i < N; ++i)
	{
		std::cout << a[i] << std::endl;
	}
	return 0;
}
```

В листинге 8 приведён пример исходного кода программы на языке C++, которая считывает с клавиатуры и сразу же выводит в консоль статический массив из десяти целых чисел. Размер массива задан в виде константы. Для ввода и вывода используются потоки C++, объявленные в заголовочном файле `iostream`. Для доступа к элементам массива индекс элементов пишется справа от переменной в квадратных скобках. Элементы массивов в C++ нумеруются с нуля.

Динамическая память работает несколько иначе, нежели статическая. Программа в любой момент может запросить у операционной системы выделить ей участок динамической памяти заданного объёма, в результате чего система пытается это сделать и сообщает в ответ адрес байта, с которого начинается выделенный участок памяти, если его действительно удалось выделить. Таким образом, можно выделять память под динамические массивы произвольного размера, в том числе вычисленного непосредственно в ходе работы программы. На момент компиляции этот размер знать необязательно.

В языке C++ динамический массив можно создать с помощью синтаксической конструкции

new тип[размер],

которая выделяет в динамической памяти массив из размер элементов типа тип и возвращает указатель на первый байт выделенной области памяти. Эта конструкция характерна именно для C++. Она отсутствовала в языке C. Также необходимо помнить, что выделенную таким образом память обязательно нужно явно освободить с помощью оператора

`delete`[] указатель,

которая принимает указатель на первый байт этой области памяти. После освобождения памяти работа с ней становится невозможной.

Листинг 9. Пример работы с динамическим массивом в языке C++

```
#include <iostream>

int main()
{
    int n;
    std::cin >> n;
    int* a = new int[n];
    for (int i = 0; i < n; ++i)
    {
        std::cin >> a[i];
    }
    for (int i = 0; i < n; ++i)
    {
        std::cout << a[i] << std::endl;
    }
    delete[] a;
    return 0;
}
```

Если забыть освободить память, а все переменные, хранящие указатель на неё, уже вышли за область своего действия, то освободить эту память в дальнейшем становится невозможным. Операционная система по-прежнему считает, что эта память занята программой, но полезная работа с ней становится невозможной. Такая ситуация называется *утечкой памяти*, а сама занятая впустую память – *мусором*. Следует избегать утечек памяти при написании программ.

В листинге 9 приведён пример исходного кода программы на языке C++, которая считывает с клавиатуры и сразу же выводит в консоль динамический массив целых чисел. Размер массива также считывается с клавиатуры в самом начале. После окончания работы с массивом выделенная под него память явным образом освобождается. Видно, что фактически массив является указателем на целое число.

В языке C динамические массивы как таковые не поддерживались, но поддерживалось выделение динамической памяти заданного объёма. Этот механизм сохранился также и в языке C++. Для выделения памяти можно использовать функцию

```
void* malloc(size_t size),
```

которая выделяет в динамической памяти участок памяти объёмом `size` байт и возвращает нетипизированный указатель на первый байт этого участка памяти. Память, выделенная таким образом, должна явно освобождаться с помощью функции

```
void free (void* ptr),
```

которая принимает указатель на первый байт участка памяти и освобождает эту память под другие нужды, после чего работа с ней становится невозможной.

Листинг 10. Пример работы с динамической памятью в языке C++

```
#include <iostream>

int main()
{
	int n;
	std::cin >> n;
	int* a = (int*)malloc(n * sizeof(int));
	for (int i = 0; i < n; ++i)
	{
		std::cin >> a[i];
	}
	for (int i = 0; i < n; ++i)
	{
		std::cout << a[i] << std::endl;
	}
	free(a);
	return 0;
}
```

В листинге 10 приведён пример использования этого старого способа работы с динамической памятью. Можно заметить, что в остальном работа с указателем не отличается от работы с массивом. Нужно лишь явно привести нетипизированный указатель к нужному типу и не забыть освободить память. Необходимый объём памяти вычисляется в программе явным образом, как произведение количества элементов в массиве на размер каждого элемента.

Этот способ работы с динамическими массивами менее наглядный, чем введённый в C++, но имеет одно преимущество. Можно изменять объём выделенной таким образом памяти прямо в ходе работы программы с помощью функции

`void* realloc (void* ptr, size_t size),`

которая принимает указатель первый байт области памяти, а также новый необходимый размер в байтах. Эта функция возвращает новый указатель, который может быть тем же самым, если операционной системе удалось перевыделить память на месте, либо другим, если ей пришлось выделить другой участок памяти нужного объёма и скопировать в него содержимое исходного участка памяти.

Листинг 11. Пример работы с классом `vector`

```
#include <iostream>
#include <vector>

int main()
{
    int n;
    std::cin >> n;
    std::vector<int> a(n);
    for (int i = 0; i < n; ++i)
    {
        std::cin >> a[i];
    }
    for (int i = 0; i < n; ++i)
    {
        std::cout << a[i] << std::endl;
    }
    return 0;
}
```

Кроме этого, в стандартной библиотеке шаблонов (STL) языка C++ в заголовочном файле `vector` объявлен удобный класс `vector`, основанный в своей реализации на работе с динамической памятью, оставшейся от языка C. Пример работы с этим классом приведён в листинге 11. Оператор «`std::vector<int> a(n)`» пред-

ставляет собой объявление переменной `a` типа `std::vector<int>`, которой при создании в конструктор передаётся количество элементов. Тип данных `std::vector<int>` по своей сути представляет собой динамический массив целых чисел.

Важным преимуществом такого подхода является отсутствие необходимости явно освобождать выделенную память, поскольку переменная `a` сама по себе создаётся на стеке, хотя и хранит внутри указатель на динамическую память. Когда программа выходит за область действия переменной `a`, эта переменная автоматически уничтожается, при этом вызывается деструктор, и вся выделенная внутри динамическая память освобождается. Кроме того, STL предоставляет широкие возможности для удобной работы с такими контейнерами, как `vector`, например, количество элементов такого вектора можно получить как `a.size()` и нет необходимости хранить отдельно. В случае достаточно глубокого понимания концепций объектно-ориентированного программирования и обобщённого программирования рекомендуется использовать класс `vector` вместо непосредственно ручной работы с массивами.

3.3 Принципы работы с массивами

Как уже было отмечено, в C++, как и во многих других языках программирования, доступ к элементам массива осуществляется с помощью операции «`[]`» путём написания индекса элемента в квадратных скобках справа от самого массива. Так можно получать значения элементов, а также присваивать им некоторые значения, так же как и обычным переменным. Важно не забывать, что элементы в массиве нумеруются с нуля.

Для единообразной обработки всех элементов массива удобно использовать итерационные циклы со счётчиком. Например, в листинге 12 показан пример функции, суммирующей элементы массива. Эта функция принимает указатель на первый элемент массива и об-

щее количество элементов и возвращает сумму элементов этого массива. Если в качестве массива используется указатель, то всегда следует хранить количество элементов в отдельной переменной, поскольку из указателя никак нельзя извлечь эту информацию.

Листинг 12. Сумма элементов массива

```
long long sum(int* a, int n)
{
    long long ans = 0;
    for (int i = 0; i < n; ++i)
    {
        ans += a[i];
    }
    return ans;
}
```

Самая популярная ошибка при работе с массивами – это выход за границы массива. *Выход за границы массива* – это ситуация, при которой происходит обращение к элементу массива с некоторым индексом, но элемент с таким индексом в массиве отсутствует. В случае нумерации элементов массива с нуля в массиве из n элементов присутствуют элементы с индексами с 0 по $(n - 1)$ включительно. Элементов с отрицательными индексами, а также с индексами n и больше в таком массиве нет.

Выход за границы массива может приводить к различным последствиям в зависимости от языка программирования и от конкретной ситуации, но всегда является ошибкой. Нужно всеми силами избегать его. В языке C++ нет внутреннего контроля выхода за границы массива, так что поведение программы в этом случае не определено. То есть будет произведено обращение к участку памяти с таким адресом, что по этому адресу должен лежать искомый элемент массива. По этому адресу может лежать какой-нибудь мусор или значение другой переменной, в этом случае просто будет произведена работа с этим участком памяти. Кроме того этот участок памяти может вообще относиться к другой программе или быть не задействован-

ным, тогда операционная система скорее всего остановит выполнение программы, пытающейся работать с участком памяти, который ей не принадлежит.

Опасность этой ошибки состоит в том, что программа может вести себя по-разному от запуска к запуску, в том числе иногда она и вовсе может работать правильно. Ошибки такого рода сложнее всего находить и исправлять. Поэтому всякий раз при обращении к элементу массива следует задумываться, не может ли индекс выйти за границы массива. Часто нужно специально делать проверку перед обращением к элементу массива и отдельно обрабатывать случай, если индекс выходит за границы.

Например, при неправильном использовании функции из листинга 12 программист может передать такой аргумент `n`, что на самом деле в этом массиве меньше `n` элементов. В этом случае в цикле произойдёт выход за границы массива, и программа может повести себя как угодно. Скорее всего, функция просто вернёт неправильное значение, но гарантировать это нельзя. В этом примере за правильностью передаваемых в функцию аргументов должен следить программист, который использует эту функцию. Часто имеет смысл выполнять некоторые проверки правильности введённых аргументов прямо внутри функции и обрабатывать ошибки определённым образом.

После создания массива, как правило, нельзя рассчитывать, что его элементы инициализированы определённым образом, так что следует заполнить его некоторыми значениями перед использованием. Например, в листингах 8, 9, 10 и 11 массив заполнялся значениями, считанными с клавиатуры. Но иногда требуется заполнить массив конкретными значениями.

Для статических массивов в C++ существует синтаксическая конструкция

тип идентификатор[размер] = {значения},

которая объявляет массив идентификатор из размер элементов типа тип и сразу же инициализирует первые элементы этого массива зна-

чениями значения, указанными через запятую. Причём размер при этом можно не указывать, тогда размер массива будет равен количеству значений в фигурных скобках. Например, конструкция

```
int a[] = {2, 3, 5, 7, 11}
```

создаёт массив `a` из пяти целых чисел: 2, 3, 5, 7 и 11, расположенных в этом порядке. А конструкция

```
int a[5] = {}
```

создаёт массив `a` из пяти нулей.

3.4 Двумерные массивы

Описанные ранее массивы называются одномерными. Многомерный массив можно понимать, как массив, элементами которого являются массивы на единицу меньшей размерности. Например, двумерный массив – это массив, элементами которого являются одномерные массивы. Двумерные массивы также называются матрицами по аналогии с математическим объектом.

Матрицы можно представлять себе как прямоугольные таблицы, в ячейках которых записаны значения элементов. Элементы матриц имеют два индекса: номер строки и номер столбца, в которых находится элемент. Следует понимать, что в программировании элементы массивов индексируются с нуля, так что номера строк и столбцов матриц также индексируются с нуля в отличие от матриц в математике.

Листинг 13. Пример работы со статическим двумерным массивом

```
int a[N][M];
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < m; ++j)
    {
        std::cin >> a[i][j];
    }
}
```

В C++ для хранения двумерных массивов существует множество способов, так же как и для хранения одномерных. Например, в листинге 13 показан пример считывания статического двумерного массива с клавиатуры. Доступ к элементам осуществляется путём написания необходимого количества индексов справа от переменной, хранящей массив, в квадратных скобках. Ещё раз следует заметить, что переменные `N` и `M` в этом коде должны быть константами, значения которых известны на момент компиляции, поскольку память под этот массив будет выделена на стеке. Причём строки такой матрицы должны храниться в памяти последовательно друг за другом, что ещё немного ускоряет последовательный доступ к элементам. Очищать память явным образом при таком подходе не требуется.

В листинге 14 приведён пример считывания динамического двумерного массива в языке C++. Массив хранится, как указатель на указатель, то есть фактически он представляет собой массив указателей на другие массивы. В данном случае переменные `n` и `m` не обязаны быть константами и могут быть предварительно считаны с клавиатуры. Однако память, выделенную таким образом под двумерный массив, нужно явным образом освободить, как и в случае одномерного массива. Участок кода, освобождающий эту память, показан в листинге 15. Видно, что сначала нужно освободить память под каждую строку матрицы, а затем уже под сам массив указателей.

Листинг 14. Считывание двумерного массива в C++

```
int** a = new int*[n];
for (int i = 0; i < n; ++i)
{
    a[i] = new int[m];
}
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < m; ++j)
    {
        std::cin >> a[i][j];
    }
}
```

Листинг 15. Освобождение памяти под двумерный массив в C++

```
for (int i = 0; i < n; ++i)
{
    delete[] a[i];
}
delete[] a;
```

Ясно, что аналогичным образом можно использовать старый способ работы с динамической памятью, оставшийся в наследство от языка C. Отличие заключается в том, что для выделения памяти следует использовать функцию `malloc` или `calloc` вместо оператора `new`, а для освобождения – функцию `free` вместо оператора `delete[]`. В остальном исходный код аналогичен листингам 14 и 15.

В листинге 16 приведён пример участка кода на языке C++, считывающего двумерный массив целых чисел, для создания которого используется контейнер `vector` из STL. Первый оператор объявляет переменную `a` типа `std::vector< std::vector<int> >`, то есть являющуюся динамическим массивом, элементами которого являются динамические массивы, элементами каждого из которых являются целые числа. Справа от переменной в круглых скобках в конструктор класса `vector` передаются аргументы: количество элементов (`n` строк) и значение для каждого элемента (`vector` из `m` целых чисел). Память, выделенную таким образом можно не освобождать явно, поскольку переменная `a` сама по себе создана на стеке. Поэтому для программистов, знакомых с объектно-ориентированным программированием и с обобщённым программированием, рекомендуется выбирать из описанных выше именно последний способ работы с двумерными массивами.

Листинг 16. Считывание матрицы, основанной на контейнере vector

```
std::vector< std::vector<int> > a(n,
std::vector<int>(m));
for (int i = 0; i < n; ++i)
{
    for (int j = 0; j < m; ++j)
    {
        std::cin >> a[i][j];
    }
}
```

3.5 Шаблонный класс vector

В этом разделе приведено чуть больше информации о классе `vector` из стандартной библиотеки шаблонов STL. Ещё раз следует отметить, что в случае недостаточного понимания таких парадигм, как объектно-ориентированное программирование и обобщённое программирование, возможно, будет лучше не использовать этот класс, а вместо этого воспользоваться более низкоуровневыми способами работы с массивами. Однако использование этого класса даёт ряд преимуществ в виде автоматической работы с памятью и наличия множества встроенных методов.

Шаблонный класс `vector` объявлен в заголовочном файле `vector` в пространстве имён `std`. Он представляет собой динамический массив, размер которого может изменяться в ходе работы программы. Параметром этого класса является тип данных элементов этого массива. Параметр шаблона указывается справа от шаблонного класса в угловых скобках. В C++ классы, предназначенные для хранения набора элементов, называются контейнерами.

По сравнению с низкоуровневыми динамическими массивами использование класса `vector` обладает следующими достоинствами.

1. Автоматическое выделение и удаление занятой памяти, ели переменная под `vector` выделена на стеке.

2. Автоматическое изменение размера при добавлении и удалении элементов, обеспечивающее быстрое добавление элементов в конец массива.

3. Встроенные методы, реализующие многие базовые алгоритмы на массивах.

4. Встроенные в библиотеку STL унифицированные средства, реализующие эффективные алгоритмы для работы с контейнерами.

5. Использование высокоуровневых библиотечных классов в исходном коде выглядит более изящно и читабельно, чем работа с низкоуровневыми динамическими массивами, особенно характерными для языка C.

Тем не менее, по сравнению с низкоуровневыми динамическими массивами класс `vector` обладает следующими несущественными недостатками.

1. Из-за особенностей выделения памяти `vector` может использовать некоторый объём дополнительной памяти впустую, не храня в ней никаких элементов.

2. Фактическая скорость работы с контейнером `vector` может быть несколько ниже, чем при работе с низкоуровневыми массивами.

3. Для понимания устройства контейнера `vector` требуется знание непростых парадигм программирования: объектно-ориентированное программирование и обобщённое программирование. Кроме того, требуется понимание того, как эти парадигмы реализованы в языке C++, в том числе знание об устройстве языка шаблонов. Всё это может вызывать лишние сложности у начинающих программистов.

Особенности контейнера `vector` по сравнению с другими контейнерами:

1. Хранит последовательность занумерованных элементов друг за другом.

2. Допускает изменение размера в ходе работы программы, особенно эффективно реализовано добавление элементов в конец контейнера.

3. Обеспечивает эффективный доступ к элементам по их номеру.

Таблица 10. Основные методы класса `vector`

Метод	**Описание**	**Сложность**
`begin()`	Итератор на начало	O(1)
`end()`	Итератор на конец	O(1)
`rbegin()`	Обратный итератор на конец	O(1)
`rend()`	Обратный итератор на начало	O(1)
`size()`	Количество элементов	O(1)
`resize(n, x)`	Делает размер равным `n`, заполняя новые элементы значением `x`	O(*n*)

`capacity()`	Количество элементов, под которые реально выделена память	O(1)
`reserve(n)`	Изменяет capacity на `n`, не изменяя формальное количество элементов	O(*n*)
`empty()`	Проверка, является ли контейнер пустым	O(1)
`at(k)`	Ссылка на `k`-ый элемент с проверкой на выход за границы массива	O(1)
`front()`	Первый элемент	O(1)
`back()`	Последний элемент	O(1)
`push_back(x)`	Добавление элемента `x` в конец	O(1)
`pop_back()`	Удаление последнего элемента	O(1)
`insert(i, x)`	Вставка элемента `x` на позицию, заданную итератором `i`	O(*n*)
`erase(i)`	Удаление элемента с позиции, заданной итератором `i`	O(*n*)
`swap(x)`	Меняется местами содержимым с вектором `x`	O(1)
`clear()`	Удаление всех элементов	O(*n*)

Ниже описаны четыре конструктора класса `vector`.

1. Конструктор по умолчанию создаёт пустой вектор.

Синтаксис: `vector<T>()`

Пример: `std::vector<int> a();`

2. Конструктор копирования создаёт копию заданного вектора.

Синтаксис: `vector<T>(const vector<T> &other)`

Пример: `std::vector<int> b(a);`

3. Конструктор, заполняющий элементы значением.

Синтаксис: `vector<T>(size_t n, const T& val = T())`

Пример: `std::vector<int> a(100000, -1);`

4. Конструктор, копирующий элементы из заданной части другого контейнера, заданной итератором на элемент, с которого надо начать копирование, и итератором на элемент, который уже не нужно включать в новый контейнер.

Синтаксис:

```
vector<T>(InputIterator first, InputIterator last)
```

Пример:

```
std::vector<int> b(a.begin(), a.begin() + 10);
```

В таблице 10 приведены основные методы класса vector. Методы записываются через точку справа от контейнера, к которому требуется применить указанной метод. Для каждого метода в таблице приводится краткое описание и вычислительная сложность. При указании вычислительной сложности под *n* по умолчанию понимается текущее количество элементов в контейнере.

3.6 Задание на лабораторную работу № 3

Вариант 0. Сумма

Программа должна складывать конечную последовательность целых чисел, введённых пользователем.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Сумма этих чисел.

Пример текстового интерфейса пользователя

```
Sum
Input the number of integers>2
Input the integer #1>2
Input the integer #2>3
Result: 5
Continue? (Y/N)>Y
Input the number of integers>3
Input the integer #1>2
Input the integer #2>3
Input the integer #3>-4
Result: 1
Continue? (Y/N)>N
```

Вариант 1. Минимум и максимум

Программа должна находить наименьшее и наибольшее число в конечной последовательности целых чисел. Наименьшее число в конечной последовательности целых чисел – это число, входящее в эту последовательность, такое, что никакое из входящих в эту последовательность чисел не меньше его. Наибольшее число в конечной последовательности целых чисел – это число, входящее в эту последовательность, такое, что никакое из входящих в эту последовательность чисел не больше его.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Наименьшее число в этой последовательности и наибольшее число в этой последовательности.

Пример текстового интерфейса пользователя

```
Minimum and maximum
Input the number of integers>2
Input the integer #1>2
Input the integer #2>3
Minimum: 2
Maximum: 3
Continue? (Y/N)>Y
Input the number of integers>3
Input the integer #1>2
Input the integer #2>3
Input the integer #3>-4
Minimum: -4
Maximum: 3
Continue? (Y/N)>N
```

Вариант 2. Наибольший общий делитель

Программа должна находить наибольший общий делитель конечной последовательности целых чисел. Наибольший общий делитель конечной последовательности целых чисел – это наибольшее целое число, на которое каждое из чисел последовательности делится без остатка.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Наибольший общий делитель этой последовательности.

Пример текстового интерфейса пользователя

```
Greatest common divisor
Input the number of integers>2
Input the integer #1>12
Input the integer #2>30
GCD = 6
Continue? (Y/N)>Y
Input the number of integers>3
Input the integer #1>24
Input the integer #2>14
Input the integer #3>35
GCD = 1
Continue? (Y/N)>N
```

Вариант 3. Удаление дубликатов

Программа должна удалять дубликаты из конечной последовательности целых чисел. Дубликат – это элемент последовательности, для которого существует равный ему элемент последовательности с меньшим номером.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Та же последовательность чисел, стоящих в том же порядке, но каждое число встречается в ней только один раз, а все последующие вхождения этого числа удалены.

Пример текстового интерфейса пользователя

```
Repeated numbers elimination
Input the number of integers>5
Input the integer #1>2
Input the integer #2>3
Input the integer #3>2
Input the integer #4>3
Input the integer #5>4
```

```
Result: 2, 3, 4
Continue? (Y/N)>Y
Input the number of integers>5
Input the integer #1>2
Input the integer #2>1
Input the integer #3>1
Input the integer #4>1
Input the integer #5>2
Result: 2, 1
Continue? (Y/N)>N
```

Вариант 4. Количество различных чисел

Программа должна вычислять количество различных чисел в конечной последовательности целых чисел.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Количество различных чисел, содержащихся в этой последовательности.

Пример текстового интерфейса пользователя

```
Count-distinct problem
Input the number of integers>5
Input the integer #1>2
Input the integer #2>3
Input the integer #3>2
Input the integer #4>3
Input the integer #5>4
Result: 3
Continue? (Y/N)>Y
Input the number of integers>5
Input the integer #1>2
Input the integer #2>1
Input the integer #3>1
Input the integer #4>1
Input the integer #5>2
Result: 2
Continue? (Y/N)>N
```

Вариант 5. Поиск представителя большинства

Программа должна находить в последовательности число, которое встречается в ней наибольшее количество раз.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Число из этой последовательности, которое встречается в ней наибольшее количество раз. Если таких чисел несколько, то можно вывести любое из них.

Пример текстового интерфейса пользователя

```
Most common element
Input the number of integers>5
Input the integer #1>2
Input the integer #2>3
Input the integer #3>3
Input the integer #4>3
Input the integer #5>2
Result: 3
Continue? (Y/N)>Y
Input the number of integers>5
Input the integer #1>2
Input the integer #2>3
Input the integer #3>2
Input the integer #4>3
Input the integer #5>2
Result: 2
Continue? (Y/N)>N
```

Вариант 6. Порядковая статистика

Назовём число первым по величине в конечной заданной последовательности целых чисел, если в этой последовательности нет чисел, меньших его. Назовём число k-ым по величине для $k > 1$, если оно больше $(k - 1)$-го по величине числа, но не существует элементов последовательности, которые меньше него и больше $(k - 1)$-го по величине числа. Программа должна находить k-е по величине число в заданной конечной последовательности целых чисел.

Входные данные. Конечная последовательность целых чисел и целое положительное число *k*.

Выходные данные. Число из этой последовательности, которое является *k*-м по величине числом этой последовательности.

Пример текстового интерфейса пользователя

```
k-th order statistics
Input k>2
Input the number of integers>5
Input the integer #1>2
Input the integer #2>4
Input the integer #3>2
Input the integer #4>3
Input the integer #5>4
Result: 3
Continue? (Y/N)>Y
Input k>3
Input the number of integers>5
Input the integer #1>5
Input the integer #2>4
Input the integer #3>3
Input the integer #4>2
Input the integer #5>4
Result: 4
Continue? (Y/N)>N
```

Вариант 7. Решето

Программа должна находить все простые числа, не превосходящие некоторого заданного натурального числа. Простое число – это целое положительное число, имеющее ровно два различных целых положительных делителя.

Входные данные. Целое положительное число.

Выходные данные. Все простые числа, не превосходящие этого целого положительного числа.

Пример текстового интерфейса пользователя

```
Prime numbers
Input an upper limit>10
Primes: 2, 3, 5, 7
Continue? (Y/N)>Y
Input an upper limit>100
Primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97
Continue? (Y/N)>N
```

Вариант 8. Считалка Иосифа

Дети встали в круг и последовательно считают друг друга по кругу, начиная с первого. Каждый *k*-ый ребёнок выбывает и круг сужается. Программа должна определять, который из детей останется последним.

Входные данные. Целое положительное изначальное количество детей и целое положительное число *k*.

Выходные данные. Изначальный номер ребёнка, который останется в круге после выбывания всех остальных детей.

Пример текстового интерфейса пользователя

```
Josephus problem
Input the number of players>10
Input k>3
Winner: 4
Continue? (Y/N)>Y
Input the number of players>5
Input k>7
Winner: 4
Continue? (Y/N)>N
```

Вариант 9. Задача о вагонах

Программа загадывает количество вагонов, входы и выходы которых последовательно соединены друг с другом: из каждого вагона можно перейти в следующий или в предыдущий, первый и последний вагоны соединены по циклу. В каждом вагоне есть лампочка, которая может быть включена, либо выключена. Начальное состояние всех

лампочек также задумывается программой. Программа должна моделировать ситуацию, в которой пользователь находится в одном из вагонов и может делать одно из трёх действий каждый ход: пойти в следующий вагон, вернуться в предыдущий вагон, изменить состояние лампочки в текущем вагоне. После каждого действия программа должна сообщать состояние лампочки в текущем вагоне. Задача пользователя – угадать количество вагонов. В любой момент пользователь может назвать ответ и программа должна его проверить.

Входные данные. Действия пользователя и ответ пользователя.

Выходные данные. Текущее состояние лампочки в текущем вагоне, а также сообщение о том, правильный ли ответ дал пользователь.

Пример текстового интерфейса пользователя

```
Count the number of railway wagons
The light is on
Input an action (next/previous/switch/answer)>next
The light is off
Input an action (next/previous/switch/answer)>next
The light is on
Input an action (next/previous/switch/answer)>switch
The light is off
Input an action (next/previous/switch/answer)>previous
The light is off
Input an action (next/previous/switch/answer)>previous
The light is off
Input an action (next/previous/switch/answer)>answer
Input the answer>2
The answer is correct
Continue? (Y/N)>Y
The light is on
Input an action (next/previous/switch/answer)>answer
Input the answer>10
The answer is incorrect
Continue? (Y/N)>N
```

Вариант 10. Сортировка индексов

Программа должна выводить номера всех элементов заданной конечной последовательности целых чисел в таком порядке, что никакой элемент, номер которого программа вывела раньше, не превосходит элемент, номер которого программа вывела позже.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Номера элементов этой последовательности в таком порядке, что элементы с такими номерами в этом порядке образуют неубывающую последовательность. Элементы в последовательности нумеруются с единицы.

Пример текстового интерфейса пользователя

```
Ascending sort of indexes
Input the number of integers>5
Input the integer #1>2
Input the integer #2>4
Input the integer #3>2
Input the integer #4>3
Input the integer #5>4
Permutation: 1, 3, 4, 2, 5
Continue? (Y/N)>Y
Input the number of integers>5
Input the integer #1>2
Input the integer #2>4
Input the integer #3>1
Input the integer #4>5
Input the integer #5>3
Permutation: 3, 1, 5, 2, 4
Continue? (Y/N)>N
```

Вариант 11. Сортировка дат

Программа должна упорядочивать заданную конечную последовательность дат в хронологическом порядке.

Входные данные. Конечная последовательность дат.

Выходные данные. Те же даты в хронологическом порядке.

Пример текстового интерфейса пользователя

```
Dates sorting
Input the number dates>5
Input the date #1 (DD.MM.YYYY)>09.05.1945
Input the date #2 (DD.MM.YYYY)>05.04.1242
Input the date #3 (DD.MM.YYYY)>08.09.1380
Input the date #4 (DD.MM.YYYY)>07.09.1812
Input the date #5 (DD.MM.YYYY)>08.07.1709
Result:
05.04.1242
08.09.1380
08.07.1709
07.09.1812
09.05.1945
Continue? (Y/N)>Y
Input the number of integers>5
Input the integer #1>20.08.1989
Input the integer #2>21.09.1988
Input the integer #3>21.07.1989
Input the integer #4>19.08.1989
Input the integer #5>19.09.1989
Result:
21.09.1988
21.07.1989
19.08.1989
20.08.1989
19.09.1989
Continue? (Y/N)>N
```

Вариант 12. Построение ломаной без самопересечений

Программа должна строить ломаную с вершинами в заданных точках на плоскости, такую что никакие два звена этой ломаной не пересекаются. Каждая заданная точка должна являться одной из вершин ломаной. Все координаты точек целочисленные.

Входные данные. Конечная последовательность точек на плоскости, заданных своими целочисленными координатами.

Выходные данные. Вершины ломаной в порядке их соединения звеньями ломаной. Звенья ломаной не должны пересекаться.

Пример текстового интерфейса пользователя

```
Simple polyline construction
Input the number of points>4
Input integer coordinates of the point #1>1 2
Input integer coordinates of the point #2>5 1
Input integer coordinates of the point #3>4 4
Input integer coordinates of the point #4>3 -1
Result:
(1, 2)
(4, 4)
(5, 1)
(3, -1)
Continue? (Y/N)>Y
Input the number of points>5
Input integer coordinates of the point #1>1 2
Input integer coordinates of the point #2>5 1
Input integer coordinates of the point #3>4 4
Input integer coordinates of the point #4>3 -1
Input integer coordinates of the point #5>3 2
Result:
(1, 2)
(4, 4)
(5, 1)
(3, -1)
(3, 2)
Continue? (Y/N)>N
```

Вариант 13. Длина объединения отрезков

Программа должна вычислять длину объединения конечного набора отрезков на координатной прямой, заданных целочисленными координатами своих концов. Точка входит в объединение отрезков, если она принадлежит хотя бы одному из них.

Входные данные. Конечная последовательность отрезков на координатной прямой, заданных целочисленными координатами своих концов.

Выходные данные. Общая длина объединения этих отрезков.

Пример текстового интерфейса пользователя

```
Length of the segments union
Input the number of segments>4
Input integer bounds of the segment #1>1 5
Input integer bounds of the segment #2>-1 2
Input integer bounds of the segment #3>3 6
Input integer bounds of the segment #4>0 4
Result: 7
Continue? (Y/N)>Y
Input the number of segments>5
Input integer bounds of the segment #1>1 3
Input integer bounds of the segment #2>2 6
Input integer bounds of the segment #3>4 5
Input integer bounds of the segment #4>7 10
Input integer bounds of the segment #5>8 9
Result: 8
Continue? (Y/N)>N
```

Вариант 14. Отсутствующее число

Программа должна выводить любое целое число, не входящее в заданную конечную последовательность целых чисел. Это число должно быть таким, что пользователь технически мог ввести его во входную последовательность.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Целое число, которое не входит в эту последовательность. Если таких чисел несколько, то можно вывести любое из них.

Пример текстового интерфейса пользователя

```
Another number
Input the number integers>4
Input the integer #1>5
Input the integer #2>2
Input the integer #3>5
Input the integer #4>3
```

```
Result: 1
Continue? (Y/N)>Y
Input the number integers>5
Input the integer #1>-1
Input the integer #2>1
Input the integer #3>-2
Input the integer #4>2
Input the integer #5>0
Result: 13
Continue? (Y/N)>N
```

Вариант 15. Суммы на отрезках

Программа должна вычислять суммы чисел на заданных пользователем отрезках заданной конечной последовательности целых чисел. Отрезок последовательности – это участок этой последовательности, состоящий из элементов этой последовательности, начиная с некоторого номера и заканчивая некоторым другим номером.

Входные данные. Конечная последовательность целых чисел и конечная последовательность отрезков.

Выходные данные. Сумма элементов на каждом введённом отрезке последовательности.

Пример текстового интерфейса пользователя

```
Sum on segments
Input the number integers>5
Input the integer #1>2
Input the integer #2>4
Input the integer #3>-1
Input the integer #4>5
Input the integer #5>1
Input bounds of the segment>2 4
Sum: 8
Continue? (Y/N)>Y
Input bounds of the segment>1 3
Sum: 5
Continue? (Y/N)>N
```

Вариант 16. Отрезок с максимальной суммой

Программа должна находить для заданной конечной последовательности целых чисел участок этой последовательности с максимальной суммой элементов.

Входные данные. Конечная последовательность целых чисел.

Выходные данные. Границы отрезка этой последовательности с максимальной суммой элементов и сама сумма элементов на этом отрезке. Если таких отрезков несколько, то можно вывести любой из них.

Пример текстового интерфейса пользователя

```
Maximum subarray
Input the number of integers>5
Input the integer #1>-1
Input the integer #2>2
Input the integer #3>4
Input the integer #4>-3
Input the integer #5>1
Maximum subarray bounds: 2 ... 3
Sum: 6
Continue? (Y/N)>Y
Input the number of integers>5
Input the integer #1>-1
Input the integer #2>2
Input the integer #3>4
Input the integer #4>-3
Input the integer #5>4
Maximum subarray bounds: 2 ... 5
Sum: 7
Continue? (Y/N)>N
```

Вариант 17. Длинная арифметика

Программа должна складывать два целых числа, возможно выходящих за рамки стандартных типов данных.

Входные данные. Два целых числа произвольной длины.

Выходные данные. Сумма этих чисел.

Пример текстового интерфейса пользователя

```
A+B for big numbers
Input an integer>2
Input an integer>3
Result: 5
Continue? (Y/N)>Y
Input an integ-
er>12963489273421694862384689273649827928
Input an integ-
er>27868236984273847289169827384791729373
Result: 408317262576955421515545166584415573 01
Continue? (Y/N)>N
```

Вариант 18. Общие числа

Программа должна находить общие числа в двух заданных конечных последовательностях целых чисел, то есть числа, содержащиеся в обеих последовательностях.

Входные данные. Две конечные последовательности целых чисел.

Выходные данные. Различные числа, входящие и в первую последовательность, и во вторую.

Пример текстового интерфейса пользователя

```
Arrays intersection
First array
Input the number of integers>5
Input the integer #1>-1
Input the integer #2>2
Input the integer #3>4
Input the integer #4>-3
Input the integer #5>1
Second array
Input the number of integers>5
Input the integer #1>5
```

```
Input the integer #2>4
Input the integer #3>3
Input the integer #4>2
Input the integer #5>1
Result: 1, 2, 4
Continue? (Y/N)>Y
First array
Input the number of integers>5
Input the integer #1>5
Input the integer #2>2
Input the integer #3>1
Input the integer #4>4
Input the integer #5>5
Second array
Input the number of integers>4
Input the integer #1>3
Input the integer #2>4
Input the integer #3>5
Input the integer #4>6
Result: 5, 4
Continue? (Y/N)>N
```

Вариант 19. Вставка и удаление элементов

Программа должна хранить конечную последовательность целых чисел и позволять пользователю последовательно вставлять элементы в эту последовательность на заданную позицию и удалять элементы этой последовательности с заданной позиции. Остальные элементы при этом должны сдвигаться и перенумеровываться. Программа должна показывать текущее состояние последовательности после каждой операции.

Входные данные. Конечная последовательность целых чисел, действия пользователя.

Выходные данные. Состояние последовательности после каждого действия пользователя.

Пример текстового интерфейса пользователя

```
Array modification
Input the number integers>5
Input the integer #1>-1
Input the integer #2>2
Input the integer #3>4
Input the integer #4>-3
Input the integer #5>1
Current array: [-1, 2, 4, -3, 1]
Input an operation (insert/erase)>insert
Input a position>3
Input an integer>10
Current array: [-1, 2, 10, 4, -3, 1]
Continue? (Y/N)>Y
Input an operation (insert/erase)>erase
Input a position>2
Current array: [-1, 10, 4, -3, 1]
Continue? (Y/N)>N
```

Вариант 20. Сортировка отрезка

Программа должна хранить конечную последовательность целых чисел и позволять пользователю упорядочивать заданный участок этой последовательности по своему выбору по неубыванию или по невозрастанию. Программа должна показывать текущее состояние последовательности после каждой операции.

Входные данные. Конечная последовательность целых чисел, границы участков, которые требуется упорядочить, и порядок сортировки.

Выходные данные. Состояние последовательности после каждого действия пользователя.

Пример текстового интерфейса пользователя

```
Array modification
Input the number integers>6
Input the integer #1>-1
Input the integer #2>0
Input the integer #3>4
Input the integer #4>-3
Input the integer #5>1
Input the integer #6>0
Current array: [-1, 0, 4, -3, 1, 0]
Input bounds of the segment>3 5
Input an order of sorting (asc/desc)>asc
Current array: [-1, 0, -3, 1, 4, 0]
Continue? (Y/N)>Y
Input bounds of the segment>2 4
Input an order of sorting (asc/desc)>desc
Current array: [-1, 1, 0, -3, 4, 0]
Continue? (Y/N)>N
```

Вариант 21. Ним

Ним – это игра для двух игроков, которые ходят по очереди. Состоянием игры являются *n* непустых кучек камней. В свой ход игрок может взять любое ненулевое количество камней из одной кучки. Выигрывает игрок, взявший последний камень. Программа должна играть в ним с пользователем. Перед игрой пользователь задаёт количество кучек и количество камней в каждой кучке, после чего пользователь и программа по очереди совершают ходы. В конце игры программа должна определять победителя.

Входные данные. Количество кучек камней, количество камней в каждой кучке, действия пользователя.

Выходные данные. Состояние игры после каждого хода, сообщение о победе одного из игроков в конце игры.

Пример текстового интерфейса пользователя

```
Nim
Input the number piles>3
Input the number of stones in the pile #1>2
Input the number of stones in the pile #2>4
Input the number of stones in the pile #3>5
Current state: [2, 4, 5]
Input the number of the pile>2
Input the number of stones to take>3
Current state: [2, 1, 5]
Bot takes 2 stones from the pile #3
Current state: [2, 1, 3]
Input the number of the pile>3
Input the number of stones to take>2
Current state: [2, 1, 1]
Bot takes 2 stones from the pile #1
Current state: [0, 1, 1]
Input the number of the pile>2
Input the number of stones to take>1
Current state: [0, 0, 1]
Bot takes 1 stones from the pile #3
Current state: [0, 0, 0]
Game over. You lose.
```

Вариант 22. Возведение матрицы в степень по модулю

Программа должна возводить заданную квадратную матрицу из целых чисел в заданную степень, причём вместо каждого элемента результирующей она должна выводить остаток от его деления на заданное натуральное число.

Входные данные. Квадратная матрица, элементами которой являются целые числа, целая неотрицательная степень, в которую её требуется возвести, и целое положительное значение модуля, по которому следует вычислить все элементы результата.

Выходные данные. Результирующая матрица.

Пример текстового интерфейса пользователя

```
Exponentiation of square matrix
Input the number of rows>2
Input the element #(1, 1)>1
Input the element #(1, 2)>2
Input the element #(2, 1)>3
Input the element #(2, 2)>4
Input a power>3
Input a modulo>100
Result:
37 54
81 18
Continue? (Y/N)>Y
Input the number of rows>3
Input the element #(1, 1)>1
Input the element #(1, 2)>2
Input the element #(1, 3)>3
Input the element #(2, 1)>4
Input the element #(2, 2)>5
Input the element #(2, 3)>6
Input the element #(3, 1)>7
Input the element #(3, 2)>8
Input the element #(3, 3)>9
Input a power>2
Input a modulo>101
Result:
30 36 42
66 81 96
1 25 49
Continue? (Y/N)>N
```

Вариант 23. Визуализация участков матрицы

Программа должна визуализировать заданные прямоугольные участки заданной прямоугольной матрицы, элементами которой являются целые числа.

Входные данные. Прямоугольная матрица, границы прямоугольных участков.

Выходные данные. Заданные участки матрицы.

Пример текстового интерфейса пользователя

```
Submatrices display
Input the number of rows>3
Input the number of columns>4
Input the element #(1, 1)>1
Input the element #(1, 2)>2
Input the element #(1, 3)>3
Input the element #(1, 4)>4
Input the element #(2, 1)>5
Input the element #(2, 2)>6
Input the element #(2, 3)>7
Input the element #(2, 4)>8
Input the element #(3, 1)>9
Input the element #(3, 2)>10
Input the element #(3, 3)>11
Input the element #(3, 4)>12
Whole matrix:
1 2 3 4
5 6 7 8
9 10 11 12
Input a left bound>1
Input a right bound>3
Input an upper bound>2
Input a lower bound>3
Result:
5 6 7
9 10 11
Continue? (Y/N)>Y
Input a left bound>2
Input a right bound>3
Input an upper bound>1
Input a lower bound>3
Result:
2 3
6 7
10 11
Continue? (Y/N)>N
```

Вариант 24. Прямоугольник из нулей

Программа должна находить в прямоугольной матрице, элементами которой являются только нули и единицы, прямоугольный участок наибольшей площади, целиком состоящий из одних нулей.

Входные данные. Прямоугольная матрица, элементами которой являются только нули и единицы.

Выходные данные. Координаты верхнего левого и правого нижнего угла самого большого по площади прямоугольного участка этой матрицы, а также площадь этого участка.

Пример текстового интерфейса пользователя

```
Largest zero-padded submatrix
Input the number of rows>4
Input the row #1>001110
Input the row #2>010000
Input the row #3>000001
Input the row #4>100100
Upper left corner: (2, 3)
Bottom right corner: (3, 5)
Total area: 6
Continue? (Y/N)>Y
Input the number of rows>5
Input the row #1>00000100
Input the row #2>00100000
Input the row #3>00000010
Input the row #4>01000000
Input the row #5>00010000
Upper left corner: (2, 4)
Bottom right corner: (4, 6)
Total area: 9
Continue? (Y/N)>N
```

Вариант 25. Поразрядная сортировка строк матрицы

Программа должна упорядочивать по неубыванию строки заданной прямоугольной матрицы, элементами которой являются целые числа. Строки упорядочиваются сначала по первому элементу, при равенстве первых элементов – по второму, и т.д. Иными словами, строка матрицы должна стоять выше другой строки, если у неё некоторый элемент строго меньше соответствующего элемента другой строки, а все элементы этой строки с меньшим номером совпадают с соответствующими элементами другой строки.

Входные данные. Прямоугольная матрица, элементами которой являются целые числа.

Выходные данные. Результирующая матрица, в которой строки исходной матрицы упорядочены по неубыванию.

Пример текстового интерфейса пользователя

```
Matrix rows sort
Input the number of rows>3
Input the number of columns>4
Input the element #(1, 1)>2
Input the element #(1, 2)>3
Input the element #(1, 3)>5
Input the element #(1, 4)>7
Input the element #(2, 1)>1
Input the element #(2, 2)>4
Input the element #(2, 3)>6
Input the element #(2, 4)>8
Input the element #(3, 1)>2
Input the element #(3, 2)>2
Input the element #(3, 3)>6
Input the element #(3, 4)>8
Initial matrix:
2 3 5 7
1 4 6 8
2 2 6 8
Sorted matrix:
1 4 6 8
2 2 6 8
2 3 5 7
```

```
Continue? (Y/N)>Y
Input the number of rows>4
Input the number of columns>3
Input the element #(1, 1)>2
Input the element #(1, 2)>4
Input the element #(1, 3)>6
Input the element #(2, 1)>3
Input the element #(2, 2)>3
Input the element #(2, 3)>5
Input the element #(3, 1)>2
Input the element #(3, 2)>4
Input the element #(3, 3)>8
Input the element #(4, 1)>2
Input the element #(4, 2)>3
Input the element #(4, 3)>7
Initial matrix:
2 4 6
3 3 5
2 4 8
2 3 7
Sorted matrix:
2 3 7
2 4 6
2 4 8
3 3 5
Continue? (Y/N)>N
```

Вариант 26. Магический квадрат

Магический квадрат – это квадратная матрица размером $n \times n$, заполненная натуральными числами от 1 до n^2 включительно, так чтобы каждое число встречалось в ней ровно один раз, а также сумма элементов в каждой строке, в каждом столбце и по обеим диагоналям была одинакова. Программа должна проверять, является ли квадратная матрица, введённая пользователем, магическим квадратом.

Входные данные. Квадратная матрица, элементами которой являются целые числа.

Выходные данные. Сообщение о том, является ли введённая матрица магическим квадратом.

Пример текстового интерфейса пользователя

```
Magic square
Input the number of rows>3
Input the element #(1, 1)>2
Input the element #(1, 2)>7
Input the element #(1, 3)>6
Input the element #(2, 1)>9
Input the element #(2, 2)>5
Input the element #(2, 3)>1
Input the element #(3, 1)>4
Input the element #(3, 2)>3
Input the element #(3, 3)>8
Whole matrix:
2 7 6
9 5 1
4 3 8
This is the magic square
Continue? (Y/N)>Y
Input the number of rows>4
Input the element #(1, 1)>7
Input the element #(1, 2)>12
Input the element #(1, 3)>1
Input the element #(1, 4)>14
Input the element #(2, 1)>2
Input the element #(2, 2)>10
Input the element #(2, 3)>3
Input the element #(2, 4)>11
Input the element #(3, 1)>16
Input the element #(3, 2)>8
Input the element #(3, 3)>13
Input the element #(3, 4)>5
Input the element #(4, 1)>9
Input the element #(4, 2)>6
Input the element #(4, 3)>15
Input the element #(4, 4)>4
Whole matrix:
7 12 1 14
2 10 3 11
16 8 13 5
9 6 15 4
This is not the magic square
Continue? (Y/N)>N
```

Вариант 27. Шидоку

Шидоку – это квадратная матрица размером 4×4, заполненная только числами 1, 2, 3 и 4, так чтобы числа в каждом столбце, в каждой строке и в каждом угловом квадрате размером 2×2 не повторялись. Программа должна достраивать введённый пользователем частично заполненный шидоку либо сообщать о том, что это невозможно.

Входные данные. Квадратная матрица размером 4×4, заполненная только числами 1, 2, 3 и 4. Некоторые элементы могут быть пропущены.

Выходные данные. Результат построения шидоку, либо сообщение о том, что построение невозможно.

Пример текстового интерфейса пользователя

```
Shidoku
Input the row #1>1234
Input the row #2>????
Input the row #3>????
Input the row #4>????
Result:
1 2 3 4
3 4 1 2
4 3 2 1
2 1 4 3
Continue? (Y/N)>Y
Input the row #1>12??
Input the row #2>3???
Input the row #3>???1
Input the row #4>????2
No solution
Continue? (Y/N)>N
```

Вариант 28. Изменение матрицы

Состоянием программы является матрица, введённая пользователем в начале её работы, элементами которой являются целые числа. Программа должна позволять пользователю изменять матрицу, вставляя и удаляя строки и столбцы по их номеру. Остальные строки и столбцы при этом должны сдвигаться. Программа должна визуализировать результат каждой операции.

Входные данные. Прямоугольная матрица, элементами которой являются целые числа, а также операции по вставке и удалению строк и столбцов.

Выходные данные. Состояние матрицы после каждой операции.

Пример текстового интерфейса пользователя

```
Matrix modification
Input the number of rows>3
Input the number of columns>4
Input the element #(1, 1)>2
Input the element #(1, 2)>3
Input the element #(1, 3)>5
Input the element #(1, 4)>7
Input the element #(2, 1)>1
Input the element #(2, 2)>4
Input the element #(2, 3)>6
Input the element #(2, 4)>8
Input the element #(3, 1)>2
Input the element #(3, 2)>2
Input the element #(3, 3)>6
Input the element #(3, 4)>8
Initial matrix:
2 3 5 7
1 4 6 8
```

```
2 2 6 8
Input the operation (insert/erase)>insert
Input the operand type (row/column)>row
Input the number of row>2
Input the row>9 8 7 6
Result:
2 3 5 7
9 8 7 6
1 4 6 8
2 2 6 8
Continue? (Y/N)>Y
Input the operation (insert/erase)>erase
Input the operand type (row/column)>column
Input the number of column>3
Result:
2 3 7
9 8 6
1 4 8
2 2 8
Continue? (Y/N)>N
```

Вариант 29. Калькулятор матриц

Состоянием программы является матрица размером 2×2, элементами которой являются дробные числа. Программа должна позволять пользователю складывать эту матрицу с другими матрицами такого же размера и умножать эту матрицу на другие матрицы такого же размера, изменяя при этом своё состояние. Программа должна визуализировать результат каждой операции.

Входные данные. Операции с текущей матрицей и квадратные матрицы из дробных чисел, с которыми нужно выполнять эти операции.

Выходные данные. Состояние матрицы после каждой операции.

Пример текстового интерфейса пользователя

```
Matrix calculator
Current state:
0.0 0.0
0.0 0.0
Input an operation (+/*)>+
Input an operand
Input the element #(1, 1)>1.5
Input the element #(1, 2)>2.5
Input the element #(2, 1)>3.5
Input the element #(2, 2)>4.5
Current state:
1.5 2.5
3.5 4.5
Continue? (Y/N)>Y
Input an operation (+/*)>*
Input an operand
Input the element #(1, 1)>5
Input the element #(1, 2)>6
Input the element #(2, 1)>7
Input the element #(2, 2)>8
Current state:
25.0 29.0
49.0 57.0
Continue? (Y/N)>N
```

Вариант 30. Крестики-нолики

Крестики-нолики играются на квадратном поле 3×3 клетки. Игроки по очереди ставят в изначально пустые клетки символы. Первый игрок ставит крестики, а второй – нолики. Выигрывает игрок, после хода которого столбец, строка или диагональ становятся полностью заполнены его символами. Программа должна играть с пользователем в крестики-нолики, позволяя ему совершать ходы и сообщая о своих ходах. Программа должна показывать состояние игрово-

го поля после каждого хода. Программа должна объявлять победителя в конце игры.

Входные данные. Ходы пользователя.

Выходные данные. Результаты ходов программы, состояние игрового поля после каждого хода и сообщение о том, кто выиграл игру.

Пример текстового интерфейса пользователя

```
Tic-tac-toe
Choose your sign (X/O)>X
Current state:
. . .
. . .
. . .
Your turn
Input the number of the row>2
Input the number of the column>2
Current state:
. . .
. X .
. . .
Bot sets 'O' at the position (1, 1)
Current state:
O . .
. X .
. . .
Your turn
Input the number of the row>1
Input the number of the column>3
Current state:
O . X
. X .
. . .
Bot sets 'O' at the position (3, 1)
Current state:
```

```
O . X
. X .
O . .
Your turn
Input the number of the row>2
Input the number of the column>3
Current state:
O . X
. X X
O . .
Bot sets 'O' at the position (2, 1)
Current state:
O . X
O X .
O . .
Game over. You lose.
```

3.7 Контрольные вопросы

1. Что такое массив и для чего он предназначен?

2. Как элементы массива хранятся в памяти компьютера?

3. Какова вычислительная сложность доступа к элементу массива по его номеру, и за счёт чего это обеспечивается?

4. Чем отличаются статические и динамические массивы?

5. Что такое утечка памяти, и каким образом образуется мусор?

6. Каким образом можно вычислить сумму всех элементов массива целых чисел? Какова вычислительная сложность такого алгоритма?

7. Что такое выход за границы массива, и как его избежать?

8. Что такое двумерные массивы и для чего они нужны?

9. Как создать двумерный массив и как освободить занятую им память?

10. Какова вычислительная сложность алгоритмов вставки элемента в начало массива и удаления элемента из начала массива?

3.8 Пример выполнения лабораторной работы

В листинге 17 приведён пример исходного кода выполненного нулевого варианта настоящей лабораторной работы. Для хранения массива используется шаблонный класс `vector`, хотя следует отметить, что в данном варианте можно было вообще обойтись без хранения массива, что позволило бы выиграть по использованию памяти, не проигрывая по быстродействию. Таким образом, формально, настоящая реализация не является наиболее эффективной и в реальности заслуживала бы некоторого снижения оценки за эффективность. Она приводится по большому счёту для лишней демонстрации принципов работы с массивами. Наконец ещё раз стоит заметить, что большая часть кода в этой программе предназначена для корректной обработки ввода данных пользователем и защиты от возможных ошибок ввода. Для успешной сдачи лабораторной работы эта часть кода не требуется: одна лишь позволяет получить более высокую оценку. Кроме того, возможно реализовать эту защиту иначе, что также может позволить сократить количество исходного кода.

Листинг 17. Пример исходного кода выполненной лабораторной работы

```
#include <iostream>
#include <string>
#include <vector>

const char EOLN = '\n';
const char YES_CHAR = 'Y';
const char NO_CHAR = 'N';
const int LEFT_BOUND = -1000000000;
const int RIGHT_BOUND = +1000000000;
const int SIZE_LIMIT = 1000000;
const std::string ABOUT_MESSAGE = "Sum";
const std::string CONTINUE_MESSAGE = "Continue? (Y/N)>";
const std::string INCORRECT_MESSAGE =
"Input is incorrect. Try again>";
const std::string INPUT_MESSAGE = "Input the integer #";
const std::string INPUT_SIZE =
```

```
"Input the number of integers>";
const std::string OUT_OF_BOUNDS_MESSAGE =
"This number is out of bounds";
const std::string OUTPUT_MESSAGE = "Result: ";
const std::string SKIP_CHARACTERS = " ";

void ClearInputStream(std::istream &in)
{
    in.clear();
    while (in.peek() != EOLN && in.peek() != EOF)
    {
        in.get();
    }
}

int Seek(std::istream &in)
{
    while (in.peek() != EOLN &&
SKIP_CHARACTERS.find((char)in.peek()) != std::string::npos)
    {
        in.get();
    }
    return in.peek();
}

bool CheckBounds(int n, int left, int right)
{
    bool ok = (left <= n && n <= right);
    if (!ok)
    {
        std::cout << OUT_OF_BOUNDS_MESSAGE << " [" << left
<< ", " << right << "]" << std::endl;
    }
    return ok;
}

int ReadInt(std::istream &in, int left, int right)
{
    int ans;
    in >> ans;
```

```
    while (!in || Seek(in) != EOLN || !CheckBounds(ans,
left, right))
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    return ans;
}

bool NeedContinue(std::istream &in)
{
    std::cout << CONTINUE_MESSAGE;
    char ans;
    in >> ans;
    while (!in || Seek(in) != EOLN || ans != YES_CHAR && ans
!= NO_CHAR)
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    return ans == YES_CHAR;
}

long long Sum(const std::vector<int> &a)
{
    long long ans = 0;
    for (int i = 0; i < (int)a.size(); ++i)
    {
        ans += a[i];
    }
    return ans;
}

int main()
{
    std::cout << ABOUT_MESSAGE << std::endl;
    bool cont = true;
    while (cont)
```

```
    {
        std::cout << INPUT_SIZE;
        int n = ReadInt(std::cin, 1, SIZE_LIMIT);
        std::vector<int> a(n);
        for (int i = 0; i < n; ++i)
        {
            std::cout << INPUT_MESSAGE << i + 1 << '>';
            a[i] = ReadInt(std::cin, LEFT_BOUND,
RIGHT_BOUND);
        }
        std::cout << OUTPUT_MESSAGE << Sum(a) << std::endl;
        cont = NeedContinue(std::cin);
    }
    return 0;
}
```

4 Обработка текстовых данных

4.1 Хранение символов в памяти компьютера

Подавляющее большинство пользовательских интерфейсов основано на обмене текстовыми данными с пользователем. Это относится даже к графическим пользовательским интерфейсам: всё равно много информации представлено в текстовом виде. Пользователь читает сообщения, нажимает на кнопки, на которых также зачастую также имеются надписи. С самого начала развития вычислительной техники придумывались способы хранения и обработки текстов в памяти компьютера.

Минимальная единица любого текста – это символ. *Символ* – это информация, соответствующая графической единице письменной речи. Буквы, цифры, знаки препинания, другие графические обозначения являются символами. Символы используются для взаимодействия с пользователем на понятном ему языке.

Символы хранятся в памяти компьютера, как обычные целые числа. Эти целые числа, которые хранятся в памяти компьютера и обозначают определённые символы, называются *кодами символов*. Отображение, связывающее коды символов с их графическим представлением называется *кодировкой*. Кодировка используется при отображении хранимых символов: вместо кодов символов отображается их графическое представление. Это происходит, например, при отображении символов на экране или при выводе символов на печать.

Существует достаточно много разнообразных кодировок. Следует понимать, что во многих случаях фактически хранятся только коды символов, а указание на то, в какой кодировке они хранятся, само по себе не хранится. Иными словами, программист должен сам понимать, в какой кодировке хранятся те или иные символы, в зависи-

мости от контекста: операционной системы, источника текста, другой метаинформации. В случае, когда символы, хранящиеся в одной кодировке, отображаются так, будто они хранятся в другой кодировке, возникают нечитаемые текстовые сообщения. Эта ошибка обычно возникает в пользовательских интерфейсах.

Наиболее исторически распространённая кодировка называется ASCII (American Standard Code for Information Interchange). Она включает символы латинского алфавита, цифры, знаки препинания, арифметические операции, а также особые управляющие символы и некоторые другие символы. В таблице 11 приведена кодировка ASCII. Номер строки означает первую цифру шестнадцатеричного кода символа, а номер столбца – вторую цифру.

Таблица 11. ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Первые 32 символа в этой таблице не имеют графических представлений, а предназначены для передачи служебной информации о тексте. Такие символы называются управляющими символами. В сиподобных языках для обозначения этих символов используются управляющие последовательности, начинающиеся с символа «\». Примеры часто используемых управляющих последовательностей: «\n» – перевод строки, «\r» – возврат каретки, «\t» – символ табуляции, «\0» – нулевой символ.

Видно, что таблица ASCII содержит 128 символов, однако в одном байте можно закодировать 256 символов. Большая часть кодировок, кодирующих каждый символ одним байтом, основаны на этом факте: первые 128 символов в них соответствуют таблице ASCII, а остальные символы отличаются. Так устроены, например, кодировки Windows-1251, KOI8-R, MacCyrillic и многие другие.

В последнее время для решения проблемы с кодировками используется кодировка Юникод, в которой один символ кодируется двумя байтами. Существует популярное представление символов в этой кодировке, называемое UTF-8, которое символы из таблицы ASCII всё равно кодирует одним байтом, но некоторые другие символы могут занимать в таком представлении более двух байт. В реальных приложениях рекомендуется придерживаться этого способа кодирования символов.

Однако в рамках лабораторных работ можно ограничиться работой с символами, которые кодируются одним байтом. В C++ для хранения таких символов используется тип данных `char`, который представляет собой обыкновенное целое число. С ними можно выполнять арифметические операции и операции сравнения. Символьные литералы в этом языке записываются в одинарных кавычках, например `'a'`, `'?'`, `'\n'`. Эти литералы также являются целыми числами, поэтому с ними можно работать, как с целыми числами.

Например, для того, чтобы проверить, хранит ли символьная переменная `c` цифру, можно воспользоваться исходным кодом

```
'0' <= c && c <= '9',
```

а для того, чтобы проверить, хранит ли эта переменная букву латинского алфавита, можно воспользоваться исходным кодом

```
'a' <= c && c <= 'z' || 'A' <= c && c <= 'Z'.
```

С другой стороны, для этих же целей предназначены оставшиеся от языка C функции `isdigit` и `isalpha`. Другой пример: для получения значения цифры по её коду можно просто вычесть из её кода код цифры 0, поскольку все цифры идут в таблице ASCII последовательно друг за другом.

4.2 Хранение строк в памяти компьютера

Строка – это конечная последовательность символов. Символы в строке имеют линейный порядок и занумерованы в соответствии с этим порядком. Количество символов в строке называется длиной строки. Понятно, что в большинстве случаев для взаимодействия с пользователем используются некоторые сообщения, представляющие собой строки, а не отдельные символы.

Строки хранятся, как массивы символов. Детальные способы хранения могут отличаться в различных языках программирования. В C++ после символов строки всегда хранится дополнительный нулевой символ с кодом 0, как это было сделано ещё в языке C. Строки, хранящиеся таким способом, называются *нуль-терминированными строками*.

Таким образом, один из способов хранения строк в языке C++ заключается в хранении динамического массива, который частично может быть заполнен символами строки, после чего следует нулевой символ. Оставшаяся часть массива после нулевого символа не считается относящейся к строке. Сам по себе нулевой символ «`\0`» не обозначает никакой символ, а как раз введён для обозначения отсутствия символа.

Чаще всего переменные, отвечающие за хранение таких строк, фактически имеют тип данных `char*`, то есть это указатели на символы. Кроме того, все строковые литералы, записывающиеся в двойных кавычках, имеют тип данных `const char*`. Например, к такому типу относятся литералы `"Hello, World!"`, `"0"` и `"\n\r"`. Стоит отметить, что управляющие последовательности можно использовать и внутри строковых литералов.

Работа непосредственно с такими динамическими массивами в C++ обычно не оправданна. Для более удобной работы со строками в STL имеется класс `string`, объектами которого являются изменяемые строки. Эти объекты в некотором роде являются контейнерами,

так что с ними можно работать, как с обычными массивами или с объектами класса `vector`. Класс `string` объявлен в одноимённом заголовочном файле `string` в пространстве имён `std`.

Листинг 18. Считывание и вывод строки в C++

```
#include <iostream>

int main()
{
    std::string s;
    std::cin >> s;
    std::cout << s << std::endl;
    return 0;
}
```

В листинге 18 приведён пример исходного кода программы на C++, которая считывает строку и тут же выводит её в консоль. При таком считывании считывается один токен, то есть последовательность печатных символов, окружённая пробелами или переводами строки. Сами пробелы при этом просто игнорируются. Для считывания строки целиком до перевода строки следует использовать функцию `getline`.

Конечно, внутри объекты класса `string` хранят обычные нультерминированные строки в виде динамических массивах, как это было принято в C. Однако STL обеспечивает богатую функциональность для работы с такими объектами как с помощью методов класса `string`, так и в виде интеграции с другими возможностями STL. На практике рекомендуется использовать этот класс для работы со строками.

4.3 Класс string

В этом разделе приведены некоторые дополнительные сведения о классе string, определённом в языке C++. Он предназначен для хранения строк, состоящих из однобайтовых символов. Как и обычно,

он не хранит кодировку, так что программисту самому следует заботиться о правильном графическом отображении хранимых символов.

В таблице 12 приведены основные методы класса `string`. Как видно, во многом они схожи с методами класса vector, описанными в таблице 10, однако имеется несколько дополнительных полезных возможностей, таких как вставка, удаление и поиск целых подстрок. Имеются обычные для STL возможности, такие как получение итераторов, запрос размера и другие. Следует отметить, что поиск подстроки в строке, как это обычно бывает в стандартных библиотеках, реализован неэффективно по вычислительной сложности. На практике для этого используются более эффективные алгоритмы.

Кроме методов класса `string` существует достаточно много переопределённых для него операций. В листинге 18 видно, что операции >> и << соответствуют чтению из потока и записи в поток, как и для многих других базовых типов данных. Операция `[]` как обычно может быть использована для обращения к символу по его номеру. Операция += нужна для добавления символов в конец строки, так же как это делает метод `push_back`.

Таблица 12. Основные методы класса `string`

Метод	**Описание**	**Сложность**
`begin()`	Итератор на начало	O(1)
`end()`	Итератор на конец	O(1)
`rbegin()`	Обратный итератор на конец	O(1)
`rend()`	Обратный итератор на начало	O(1)
`size()`	Длина строки	O(1)
`max_size()`	Максимальная длина строки	O(1)
`resize(n, x)`	Делает размер равным `n`, заполняя новые элементы символом `x`	O(*n*)
`capacity()`	Количество символов, под которые реально выделена память	O(1)
`reserve(n)`	Изменяет capacity на `n`, не изменяя формальное количество символов	O(*n*)
`clear()`	Удаление всех элементов	O(*n*)

`empty()`	Проверка на пустую строку	O(1)
`at(k)`	Ссылка на `k`-ый элемент с проверкой на выход за границы массива	O(1)
`push_back(x)`	Добавление символа `x` в конец	O(1)
`insert(i, x)`	Вставка подстроки `x` на позицию `i`	O(n)
`erase(i, l)`	Удаление подстроки длины `l`, начиная с позиции `i`	O(n)
`swap(x)`	Меняется местами содержимым со строкой `x`	O(1)
`c_str()`	Нуль-терминированная строка	O(1)
`find(x)`	Поиск первого вхождения подстроки `x` длины m	O(nm)
`rfind(x)`	Поиск последнего вхождения подстроки `x` длины m	O(nm)
`substr(i, n)`	Подстрока длины `n`, начинающаяся с позиции `i`	O(n)

Также две строки можно сложить с помощью операции +. В результате получается строка, представляющая собой конкатенацию операндов, то есть состоящая сначала из символов первой строки, а затем из символов второй строки. Конкатенация строки длины n и строки длины m выполняется за O($n + m$) операций.

Для типа `string` определён лексикографический порядок, позволяющий сравнивать строки с помощью операций <, >, <=, >=. В лексикографическом порядке пустая строка предшествует непустой, а две непустые строки сравниваются по первому символу: у какой из них код первого символа меньше, та и предшествует другой. Если же первые символы двух непустых строк одинаковы, то по тем же правилам сравниваются подстроки этих строк без первых символов. Иными словами, лексикографический порядок похож на порядок, в котором слова располагаются в словаре: сначала по первой букве, потом по второй и так далее.

Для преобразования динамического массива символов, хранящего нуль-терминированную строку, можно использовать обычный оператор присваивания или передать эту строку в конструктор. Для

обратного преобразования, как можно судить из таблицы 12, используется метод `c_str`. Дополнительным преимуществом при использовании объектов класса `string` является автоматическая работа с памятью, что особенно удобно, если переменные типа `string` создаются на стеке.

4.4 Работа с файлами

Файл – это именованная область данных. Обычно файлы хранятся на внешних носителях, но также могут содержаться в оперативной памяти, являться по своей сути потоками или обеспечивать доступ к определённым устройствам. Способ организации и хранения файлов называется *файловой системой*. Работа с файлами осуществляется средствами операционной системы.

С точки зрения программиста файлы – это внешние ресурсы, предназначенные для ввода и вывода. Работа с файлами всегда строится на трёх последовательных шагах.

1. Открытие файла. При этом операционная система предоставляет программе доступ к файлу, если это возможно, а также частично ограничивает доступ к этому файлу для других программ.

2. Ввод из файла или вывод в файл. При этом можно выяснить содержимое файла или наоборот записать в файл некоторую информацию.

3. Закрытие файла. При этом операционная система понимает, что файл больше не используется, и разблокирует доступ к нему для других программ.

Листинг 19. Считывание и вывод строки в C++

```
#include <fstream>
#include <string>

int main()
{
    std::ifstream fin("input.txt");
```

```
    std::string s("");
    if (fin.is_open())
    {
        char c;
        while (fin.get(c))
        {
            s += c;
        }
        fin.close();
    }
    std::ofstream fout("output.txt");
    if (fout.is_open())
    {
        fout << s;
        fout.close();
    }
    return 0;
}
```

В C++ для работы с файлами удобно использовать файловые потоки, объявленные в стандартном заголовочном файле `fstream` в пространстве имён `std`. Для ввода данных из файла используется поток класса `ifstream`, а для вывода данных в файл – поток класса `ofstream`. Работа с этими потоками осуществляется точно так же, как и с консольными потоками `cin` и `cout`. Для ввода и вывода отдельных токенов используются соответственно операции >> и <<, для ввода и вывода отдельного символа можно использовать соответствовать методы `get` и `put`. Для ввода строки целиком до перевода строки используется функция `getline`, объявленная в заголовочном файле `string`. Для вывода перевода строки можно использовать объект `endl`, объявленный в заголовочном файле `iostream`.

В листинге 19 приведён пример исходного кода программы на C++, которая посимвольно вычитывает содержимое файла `input.txt`, после чего записывает прочитанное в файл `output.txt`.

Названия файлов передаются в конструктор потоков при создании. В данном случае это относительные имена, так что будет произведена работа с файлами в текущей директории, но также можно использовать и абсолютные имена. Метод `is_open` используется для проверки, удалось ли открыть файл. После работы с файлом он явным образом закрывается путём вызова метода `close`.

При работе с файлами, как и с другими внешними ресурсами, нужно стараться как можно меньше времени держать их открытыми, ведь другие программы также могут нуждаться в работе с этими ресурсами. Типичная ошибка при работе с файлами – отсутствие операции закрытия файла, так что файл остаётся открытым даже когда это уже не требуется. При этом он остаётся заблокированным и работа с ним остаётся невозможной. Когда пользователь пытается изменить этот файл из другого приложения, операционная система сообщает ему об ошибке. Таких ситуаций следует избегать. Нужно как можно скорее после открытия файла выполнять все необходимые операции чтения или записи с ним, после чего сразу же его закрывать.

4.5 Задание на лабораторную работу № 4

Вариант 0. Эхо

Программа должна выводить строку, введённую пользователем.

Входные данные. Строка печатных символов, заканчивающаяся переводом строки.

Выходные данные. Эта же строка.

Пример текстового интерфейса пользователя

```
Echo
Input a string>Hello, World!
Hello, World!
Continue? (Y/N)>Y
Input a string>'; DROP TABLE users;
'; DROP TABLE users;
Continue? (Y/N)>N
```

Вариант 1. Поиск в текстовом файле

Программа должна находить в заданном текстовом файле первую строку, в которой встречается заданная подстрока. Считается, что текстовый файл состоит из строк, оканчивающихся переводами строки.

Входные данные. Текстовый файл, заданный своим именем, и строка из печатных символов.

Выходные данные. Номер первой строки в этом текстовом файле, содержащей входную строку в качестве подстроки, а также сама эта строка.

Пример текстового интерфейса пользователя

```
Text search
Input a string>Wolf
Input a filename>The Jungle Book.txt
Line #41:
Father Wolf woke up from his day's rest, scratched
himself, yawned, and
Continue? (Y/N)>Y
Input a string> woman
Input a filename> The Adventures of Sherlock Holmes.txt
Line #29:
To Sherlock Holmes she is always THE woman. I have
seldom heard
Continue? (Y/N)>N
```

Вариант 2. Наиболее популярное слово

Программа должна находить в заданном текстовом файле, содержащем текст на естественном английском языке, слово, которое встречается в нём чаще всего. Слова в таком файле состоят из подряд идущих больших и малых символов латинского алфавита и разделяются любыми другими символами. Слова, отличающиеся только регистром символов, следует считать одинаковыми.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Слово, встречающееся в этом файле чаще всего. Если таких слов несколько, то можно вывести любое из них.

Пример текстового интерфейса пользователя

```
Most popular word
Input a filename>The Jungle Book.txt
Result: the
Continue? (Y/N)>Y
Input a filename> The Adventures of Sherlock Holmes.txt
Result: the
Continue? (Y/N)>N
```

Вариант 3. Уникальные слова

Программа должна находить в заданном текстовом файле, содержащем текст на естественном английском языке, уникальные слова, то есть слова, встречающиеся в этом файле лишь один раз. Слова в таком файле состоят из подряд идущих больших и малых символов латинского алфавита и разделяются любыми другими символами. Слова, отличающиеся только регистром символов, следует считать одинаковыми.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Все слова, встречающиеся в этом файле ровно один раз.

Пример текстового интерфейса пользователя

```
Unique words
Input a filename>The Jungle Book.txt
Result: guards, result, overtaking, ...
Continue? (Y/N)>Y
Input a filename> The Adventures of Sherlock Holmes.txt
Result: glade, imbedded, sunset, ...
Continue? (Y/N)>N
```

Примечание. В примере вывод программы сокращён.

Вариант 4. Частота символов

Программа должна для каждого символа, встречающегося в заданном текстовом файле, выводить частоту, с которой этот символ встречается в этом файле. Частота, с которой символ встречается в текстовом файле, – это отношение количества таких символов в этом файле к общему количеству символов в этом файле.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Частота, с которой каждый символ, встречающийся в этом файле, встречается в этом файле.

Пример текстового интерфейса пользователя

```
Letter frequency
Input a filename>The Jungle Book.txt
'A': 0.00143989007134
'B': 0.00228043503684
'C': 0.000588381475851
...
Continue? (Y/N)>Y
Input a filename> The Adventures of Sherlock Holmes.txt
'A': 0.00135325216191
'B': 0.000851784212295
'C': 0.000581489430941
...
Continue? (Y/N)>N
```

Примечание. В примере вывод программы сокращён.

Вариант 5. Архивирование текста

Программа должна архивировать заданный достаточно большой текстовый файл, содержащий текст на естественном английском языке, в файл меньшего объёма, а также разархивировать его обратно.

Входные данные. Команда архивирования либо разархивирования, а также имя исходного файла и имя файла с результатом операции.

Выходные данные. Файл с результатом архивирования либо разархивирования.

Пример текстового интерфейса пользователя

```
File archiver
Input a command (zip/unzip)>zip
Input a source filename>The Jungle Book.txt
Input a destination filename>The Jungle Book.arc
Operation has been completed successfully
Continue? (Y/N)>Y
Input a command (zip/unzip)>unzip
Input a source filename>The Jungle Book.arc
Input a destination filename>output.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 6. Шифрование текста

Программа должна зашифровывать заданный текстовый файл, содержащий текст на естественном английском языке, и записывать результат в другой файл, а также расшифровывать его обратно.

Входные данные. Команда зашифровывания либо расшифровывания, а также имя исходного файла и имя файла с результатом операции.

Выходные данные. Файл с результатом зашифровывания либо расшифровывания.

Пример текстового интерфейса пользователя

```
Cryptor
Input a command (encrypt/decrypt)>encrypt
Input a source filename>The Jungle Book.txt
Input a destination filename>The Jungle Book.enc
Operation has been completed successfully
Continue? (Y/N)>Y
Input a command (encrypt/decrypt)>decrypt
Input a source filename>The Jungle Book.enc
Input a destination filename>output.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 7. Типографские правки

Программа должна выполнять следующие изменения в заданном файле, содержащем текст на естественном английском языке.

1. Несколько пробелов подряд заменить на один пробел.

2. Перед точкой, запятой, восклицательным и вопросительным знаками, а также перед закрывающими скобками удалить пробелы.

3. После точки, запятой, восклицательного и вопросительного знаков, а также после закрывающей скобки поставить пробелы, если их там нет.

4. После открывающей скобки удалить пробелы.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Typographical changes
Input a filename>The Jungle Book.txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 8. Удаление дубликатов

Программа должна в заданном текстовом файле, содержащем текст на естественном английском языке, заменять серию из подряд идущих одинаковых слов, разделённых пробелами, на одно такое слово. Слова в таком файле состоят из подряд идущих больших и малых символов латинского алфавита и разделяются любыми другими символами. Слова, отличающиеся только регистром символов, следует считать одинаковыми.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Duplicates removal
Input a filename>The Jungle Book.txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 9. Римские цифры

Программа должна в заданном текстовом файле, содержащем текст на естественном английском языке, заменять римские числа на их арабские аналоги. Римские числа состоят из подряд идущих символов «I», «V», «X», «L», «C», «D», «M», отделённых от слов и других чисел символами, не являющимися ни большими, ни малыми латинскими буквами. Значения римских цифр: «I» – 1, «V» – 5, «X» – 10, «L» – 50, «C» – 100, «D» – 500, «M» – 1000. Числа, записанные римскими цифрами, получаются сложением входящих в них цифр за исключением случая, когда меньшая цифра в числе стоит перед большей цифрой: в этом случае эта цифра вычитается, а не прибавляется.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Roman numerals
Input a filename>The Jungle Book.txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 10. Капитализация

Программа должна в заданном текстовом файле, содержащем текст на естественном английском языке, заменить первую букву в каждом слове на соответствующую заглавную букву. Слова в таком файле состоят из подряд идущих больших и малых символов латинского алфавита и разделяются любыми другими символами.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Capitalization
Input a filename>The Jungle Book.txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 11. Разбиение на строки

Программа должна разбивать текст в текстовом файле, содержащем текст на естественном английском языке, на строки, добавляя переводы строки, так чтобы количество символов в каждой строке не превышало некоторого заданного числа. При этом слова из исходного файла не должны разделяться на части. Словами в таком файле можно считать последовательности подряд идущих печатных символов, разделённых пробелами и переводами строки.

Входные данные. Текстовый файл, заданный своим именем, а также целое положительное число, представляющее собой ограничение на длину строки. Файл должен содержать текст на естественном английском языке.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Text Justification
Input a filename>The Jungle Book.txt
Input a line width>80
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Input a line width>42
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 12. Проверка правильности расстановки скобок

Программа должна проверять, правильно ли в текстовом файле расставлены круглые скобки. Будем считать, что если в тексте нет круглых скобок, то они расставлены правильно. Если текст, в котором правильно расставлены скобки, взять в круглые скобки, то в получившемся тексте скобки также будут расставлены правильно. Кроме того, конкатенация двух текстов, в которых правильно расставлены скобки, также является текстом, в котором правильно расставлены скобки. В текстах, не удовлетворяющих этим правилам, скобки расставлены неправильно.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Сообщение о том, правильно ли расставлены скобки в этом файле.

Пример текстового интерфейса пользователя

```
Parentheses Balance Check
Input a filename>The Jungle Book.txt
Parentheses are balanced
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Parentheses are balanced
Continue? (Y/N)>N
```

Вариант 13. Статистика текста

Программа должна вычислять количество слов, общее количество символов и количество символов, исключая пробелы, в текстовом файле, содержащем текст на естественном английском языке. Слова в таком файле состоят из подряд идущих больших и малых символов латинского алфавита и разделяются любыми другими символами.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Количество слов в этом файле, общее количество символов и количество символов, исключая пробелы.

Пример текстового интерфейса пользователя

```
Text statistics
Input a filename>The Jungle Book.txt
Words: 51146
Characters (no spaces): 219517
Characters (with spaces): 268220
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Words: 104685
Characters (no spaces): 455000
Characters (with spaces): 549683
Continue? (Y/N)>N
```

Вариант 14. Гласные и согласные буквы

Программа должна вычислять количество гласных и согласных латинских букв в текстовом файле. Гласными буквами считаются буквы «A», «E», «I», «O», «U», «Y». Остальные латинские буквы считаются согласными. Буквы, отличающиеся только регистром, считаются одной и той же буквой.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Количество гласных латинских букв в этом файле и количество согласных латинских букв в этом файле.

Пример текстового интерфейса пользователя

```
Vowels and Consonants
Input a filename>The Jungle Book.txt
Vowels: 81262
Consonants: 126928
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Vowels: 174432
Consonants: 257302
Continue? (Y/N)>N
```

Вариант 15. Количество предложений

Программа должна вычислять количество предложений в текстовом файле, содержащем текст на естественном английском языке. Предложением считается непустая последовательность печатных символов, оканчивающаяся точкой, вопросительным знаком или восклицательным знаком.

Входные данные. Текстовый файл, заданный своим именем. Файл должен содержать текст на естественном английском языке.

Выходные данные. Количество предложений в этом файле.

Пример текстового интерфейса пользователя

```
Sentences count
Input a filename>The Jungle Book.txt
Number of sentences: 3214
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Number of sentences: 7288
Continue? (Y/N)>N
```

Вариант 16. Поиск чисел в тексте

Программа должна находить все числа, встречающиеся в текстовом файле. Числами можно считать последовательности из подряд идущих арабских цифр, отделённых друг от друга любыми другими символами.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Числа, встречающиеся в этом текстовом файле, в том порядке, в котором они встречаются.

Пример текстового интерфейса пользователя

```
Number detector
Input a filename>The Jungle Book.txt
Numbers: 1842, 39, 39
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Numbers: 1888, 1858, 5, ...
Continue? (Y/N)>N
```

Примечание. В примере вывод программы сокращён.

Вариант 17. Поиск наиболее ранней даты в тексте

Программа должна находить наиболее раннюю дату, встречающуюся в текстовом файле. Даты записываются в формате «ДД.ММ.ГГГГ», где «ДД» – это число, «ММ» – это номер месяца, «ГГГГ» – это год, и разделяются любыми другими символами.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Хронологически наиболее ранняя дата из встречающихся в этом файле.

Пример текстового интерфейса пользователя

```
Earliest date
Input a filename>The Jungle Book.txt
There are no dates in format "DD.MM.YYYY"
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
There are no dates in format "DD.MM.YYYY"
Continue? (Y/N)>N
```

Вариант 18. Сортировка строк

Программа должна переставлять местами строки в текстовом файле, так чтобы они шли в лексикографическом порядке. Строки в текстовом файле – это последовательности символов, заканчивающиеся переводом строки. Пустая строка предшествует любой непус-

той строке в лексикографическом порядке. Из двух непустых строк предшествует та, у которой меньше код первого символа. Если у двух непустых строк одинаковые первые символы, то у них такой же порядок, как и у их соответствующих подстрок без первого символа.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Lines sort
Input a filename>The Jungle Book.txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 19. Цензура

Программа должна заменять заданную подстроку в заданном текстовом файле на другую заданную подстроку.

Входные данные. Текстовый файл, заданный своим именем.

Выходные данные. Тот же файл с необходимыми изменениями.

Пример текстового интерфейса пользователя

```
Censorship
Input a filename>The Jungle Book.txt
Input a string to find> Shere Khan
Input a string to replace with>Lord of the Jungle
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Input a string to find> Moriarty
Input a string to replace with>Anonymous
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 20. Наибольшая общая подстрока

Программа должна находить наибольшую общую подстроку в двух заданных текстовых файлах. Общая подстрока для этих двух файлов – это подстрока, встречающаяся как в первом файле, так и во втором. Наибольшая общая подстрока – это самая длинная из общих подстрок.

Входные данные. Два текстовых файла, заданных своим именем.

Выходные данные. Наибольшая общая подстрока для этих двух файлов. Если таких строк несколько, то можно вывести любую из них.

Пример текстового интерфейса пользователя

```
Longest common substring
Input a filename>The Jungle Book.txt
Input a filename>The Adventures of Sherlock Holmes.txt
Result: '
                                                      '
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Input a filename>Jane Eyre.txt
Result: ' had something else to think '
Continue? (Y/N)>N
```

Вариант 21. Обмен

Программа должна менять местами содержимое двух заданных текстовых файлов.

Входные данные. Два текстовых файла, заданных своим именем.

Выходные данные. Те же файлы с изменённым содержимым.

Пример текстового интерфейса пользователя

```
Swap files
Input a filename>The Jungle Book.txt
Input a filename>The Adventures of Sherlock Holmes.txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a filename>The Adventures of Sherlock Holmes.txt
Input a filename>Jane Eyre.txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 22. Похожие файлы

Программа должна определять, можно ли так поменять местами символы одного заданного текстового файла, чтобы получить содержимое другого заданного текстового файла.

Входные данные. Два текстовых файла, заданных своим именем.

Выходные данные. Сообщение о том, можно ли поменять местами некоторое произвольное количество символов в первом файле, чтобы он стал копией второго файла.

Пример текстового интерфейса пользователя

```
File similarity check
Input a filename>The Jungle Book.txt
Input a filename>The Adventures of Sherlock Holmes.txt
It's impossible to rearrange characters in the first
file to get the second file
Continue? (Y/N)>Y
Input a filename>Jane Eyre.txt
Input a filename>Jane Eyre.txt
It's possible to rearrange characters in the first
file to get the second file
Continue? (Y/N)>N
```

Вариант 23. Волшебная цепочка

Программа должна определять, можно ли получить одно заданное слово из другого заданного слова, каждый раз изменяя в нём по одной букве, так чтобы каждый раз получалось одно из слов, содержащихся в заданном текстовом файле. Слова состоят только из больших и малых букв латинского алфавита. Слова в файле записаны по одному в строке. Каждая строка завершается переводом строки.

Входные данные. Исходное слово, конечное слово и текстовый файл со словами, заданный своим именем. Слова в текстовом файле расположены по одному в строке.

Выходные данные. Цепочка слов, содержащихся в текстовом файле, начинающаяся и заканчивающаяся на заданные слова, такая что соседние слова в ней отличаются одной буквой, либо сообщение о том, что такую цепочку построить невозможно.

Пример текстового интерфейса пользователя

```
Word Morph
Input a dictionary filename> nounlist.txt
Input a source word>geek
Input a destination word>dork
No solution
Continue? (Y/N)>Y
Input a filename>boat
Input a filename>duck
boat -> boot -> book -> look -> lock -> dock -> duck
Continue? (Y/N)>N
```

Вариант 24. Копирование файлов

Программа должна создавать новый файл с заданным именем, содержимое которого полностью совпадает с содержимым другого файла с заданным именем.

Входные данные. Исходный файл, заданный своим именем, а также имя нового файла.

Выходные данные. Копия исходного файла с заданным именем.

Пример текстового интерфейса пользователя

```
File copy
Input a source filename>The Jungle Book.txt
Input a destination filename>The Jungle Book (1).txt
Operation has been completed successfully
Continue? (Y/N)>Y
Input a source filename>Jane Eyre.txt
Input a destination filename>Jane Eyre (1).txt
Operation has been completed successfully
Continue? (Y/N)>N
```

Вариант 25. Текстовый редактор

Программа должна выводить заданные строки заданного текстового файла по их номеру, а также менять их на другие строки, введённые пользователем. Строки в текстовом файле заканчиваются символом перевода строки.

Входные данные. Текстовый файл, заданный своим именем, а также операции визуализации строки по номеру и замены строки с заданным номером на другую строку.

Выходные данные. Результаты операций.

Пример текстового интерфейса пользователя

```
Text editor
Input a filename>The Jungle Book.txt
Input an operation (view/edit/exit)>view
Input the number of the line>40
It was seven o'clock of a very warm evening in the
Seeonee hills when
Input an operation (view/edit/exit)>edit
Input the number of the line>40
Input a line to replace with>abacaba
Operation has been completed successfully
Input an operation (view/edit/exit)>view
Input the number of the line>40
abacaba
Input an operation (view/edit/exit)>exit
```

Вариант 26. Персистентный калькулятор целых чисел

Состоянием программы является целое число. Пользователь может выполнять с этим числом арифметические операции: сложение, вычитание, умножение и целочисленное деление, изменяя состояние. Программа должна визуализировать результат каждой операции. Также пользователь может восстановить состояние программы, которое было заданное количество операций назад. Программа должна сохранять состояние между отключениями компьютера.

Входные данные. Операции и целые операнды.

Выходные данные. Результат каждой операции.

Пример текстового интерфейса пользователя

```
Persistent integer calculator
Current state: 0
Input an operation (+/-/*///load/exit)>+
Input an operand>7
Current state: 7
Input an operation (+/-/*///load/exit)>/
Input an operand>3
Current state: 2
Input an operation (+/-/*///load/exit)>load
Input the number of operations to undo>1
Current state: 7
Input an operation (+/-/*///load/exit)>exit
```

Вариант 27. Шашечная доска

Программа должна определять, какие из шашек находятся под боем, по заданному описанию шашечной доски, содержащемуся в заданном текстовом файле. Файл с описанием шашечной доски содержит 8 строк по 8 символов в каждой строке. В i-ой строке на j-ой позиции содержится описание соответствующей клетки шашечной доски с координатами (i, j): символ «`.`» означает, что в клетке отсутствует шашка, символ «`w`» означает, что в клетке присутствует белая шашка, а символ «`b`» означает, что в клетке присутствует чёрная шашка. В шашках за один ход шашка одного цвета может взять шашку другого цвета, если она является соседней с ней по диагонали (то есть каждая её координата отличается ровно на единицу), и есть возможность перепрыгнуть эту шашку, изменив каждую свою координату на 2 единицы в направлении шашки, которую планируется взять. Это невозможно, если в той клетке уже стоит любая другая шашка. Взятая шашка немедленно снимается с поля. Если шашка берёт другую шашку, она в этот же ход может взять ещё шашку по тем же правилам, пока у неё есть такая возможность. Считается, что шашка под боем, если её можно взять за один ход.

Входные данные. Файл с описанием шашечной доски, заданный своим именем.

Выходные данные. Координаты шашек, находящихся под боем.

Пример текстового интерфейса пользователя

```
Checkers position analyzer
Input a filename>board-01.txt
Pieces under attack: (2, 2), (4, 6)
Continue? (Y/N)>Y
Input a filename>board-02.txt
No pieces are under attack
Continue? (Y/N)>N
```

Вариант 28. Игра в слова

Два игрока играют в слова. Они по очереди называют слова, содержащиеся в заданном текстовом файле, так чтобы каждое следующее слово начиналось на ту букву, на которую заканчивалось предыдущее слово. В самом первом ходу можно назвать любое из слов, содержащихся в текстовом файле. Программа должна играть с игроком в слова, называя корректные слова и проверяя ходы игрока. Если игрок называет слово не из текстового файла, либо начинающееся не на ту букву, на которую закончилось предыдущее слово, то он проигрывает, и программа должна сообщить ему об этом. Слова в текстовом файле записаны по одному в строке и состоят только из малых букв латинского алфавита.

Входные данные. Текстовый файл со словами, заданный своим именем, а также ходы пользователя.

Выходные данные. Ходы программы и сообщение о некорректном ходе пользователя.

Пример текстового интерфейса пользователя

```
Word chain
Input a filename>nounlist.txt
Input a word>program
Bot says: main
Input a word>node
Bot says: end
Input a word>begin
Incorrect
Game over. You lose.
```

Вариант 29. Быки и коровы

Программа должна играть с пользователем в упрощённый вариант игры «Быки и коровы». Программа задумывает одно из слов, содержащихся в текстовом файле, и сообщает пользователю количество букв в нём, а пользователь должен отгадать это слово. Каждый ход пользователь называет одно из слов заявленной длины, содержащихся в текстовом файле, а программа сообщает количество букв в названном слове, совпадающих с буквами в загаданном слове, стоящими на тех же позициях. Когда пользователь называет загаданное слово, программа должна сообщить об этом. Слова в текстовом файле записаны по одному в строке.

Входные данные. Текстовый файл со словами, заданный своим именем, а также ходы пользователя. Слова в текстовом файле записаны по одному в строке.

Выходные данные. Сообщения о количестве совпавших букв, а также сообщение о конце игры.

Пример текстового интерфейса пользователя

```
Bulls and Cows
Input a filename>nounlist.txt
Word length: 4
Input a word>boat
The number of matches: 0
Input a word>park
The number of matches: 2
Input a word>pass
The number of matches: 4
Game over. You win.
```

Вариант 30. Лабиринт

Программа должна моделировать перемещения пользователя по лабиринту, карта которого содержится в текстовом файле. Лабиринт представляет собой клетчатое поле, некоторые клетки которого являются проходами, а некоторые – стенами. Вначале пользователь помещается в случайный проход. Он может перемещаться в одном из

четырёх направлений по одной клетке, если в этом направлении существует соседняя клетка, являющаяся проходом. Программа должна сообщать пользователю, удалось ли ему переместиться в желаемом направлении. Если пользователю удалось выбраться за границы клетчатого поля, то считается, что он покинул лабиринт и выиграл. В этом случае программа должна сообщить ему об этом.

Входные данные. Текстовый файл с картой лабиринта, заданный своим именем, а также ходы пользователя. Карта лабиринта в первой строке содержит два положительных целых числа *n* и *m* – количество строк и количество столбцов клетчатого поля. Далее в *n* строках содержатся по *m* символов – информация о клетках. В *i*-й строке на *j*-й позиции содержится информация о клетке с координатами (*i*, *j*). Символ «**.**» означает проход, а символ «#» – стену.

Выходные данные. Сообщения о том, удалось ли пользователю выполнить желаемый ход, а сообщение о том, что пользователю удалось выбраться из лабиринта.

Пример текстового интерфейса пользователя

```
Labyrinth
Input a filename>map.txt
Input a direction (left/right/up/down)>left
You have passed
Input a direction (left/right/up/down)>left
A wall blocks the passage
Input a direction (left/right/up/down)>up
You have passed
Input a direction (left/right/up/down)>left
You have passed
Input a direction (left/right/up/down)>down
You have passed
Input a direction (left/right/up/down)>left
You have successfully left the labyrinth
```

4.6 Контрольные вопросы

1. Что такое символ в программировании? Каким образом символы хранятся в памяти компьютера?
2. Что такое кодировка? Для чего она предназначена?
3. Какие распространённые кодировки вы знаете? Чем они отличаются?
4. Что такое таблица ASCII? Какие символы в неё входят?
5. Что такое Юникод? Сколько байт отводится на хранение одного символа в представлении UTF-8?
6. Что такое строка в программировании? Для чего она предназначена?
7. Как строки хранятся в памяти компьютера? Чем могут отличаться различные способы хранения строк?
8. Что такое файл? Для чего он предназначен?
9. Что такое файловая система? Для чего она предназначена?
10. Каков порядок работы с файлами в программировании? Для чего нужна операция закрытия файла?

4.7 Пример выполнения лабораторной работы

В листинге 20 приведён пример исходного кода выполненной лабораторной работы на языке C++. Большая часть проверок на ввод осуществляется только для считывания ответа на запрос о продолжении работы программы, так что без них можно было бы обойтись, убрав эту функциональность из программы. Для успешной сдачи лабораторной работы достаточно просто считывать строку, выводить её и завершать работу программы.

Листинг 20. Пример исходного кода выполненной лабораторной работы

```
#include <iostream>
#include <string>

const char EOLN = '\n';
const char YES_CHAR = 'Y';
const char NO_CHAR = 'N';
```

```
const std::string ABOUT_MESSAGE = "Echo";
const std::string CONTINUE_MESSAGE = "Continue? (Y/N)>";
const std::string INCORRECT_MESSAGE =
"Input is incorrect. Try again>";
const std::string INPUT_MESSAGE = "Input a string>";
const std::string SKIP_CHARACTERS = " ";

void ClearInputStream(std::istream &in)
{
    in.clear();
    while (in.peek() != EOLN && in.peek() != EOF)
    {
        in.get();
    }
}

int Seek(std::istream &in)
{
    while (in.peek() != EOLN &&
SKIP_CHARACTERS.find((char)in.peek()) != std::string::npos)
    {
        in.get();
    }
    return in.peek();
}

bool NeedContinue(std::istream &in)
{
    std::cout << CONTINUE_MESSAGE;
    char ans;
    in >> ans;
    while (!in || Seek(in) != EOLN || ans != YES_CHAR && ans
!= NO_CHAR)
    {
        ClearInputStream(in);
        std::cout << INCORRECT_MESSAGE;
        in >> ans;
    }
    if (in.peek() == EOLN)
    {
```

```
        in.get();
    }
    return ans == YES_CHAR;
}

int main()
{
    std::cout << ABOUT_MESSAGE << std::endl;
    bool cont = true;
    while (cont)
    {
        std::cout << INPUT_MESSAGE;
        std::string s;
        std::getline(std::cin, s);
        std::cout << s << std::endl;
        cont = NeedContinue(std::cin);
    }
    return 0;
}
```

Заключение

В настоящем издании предлагается ряд заданий для самостоятельного выполнения лабораторных работ по курсу «Основы информатики». Все они связаны с написанием программ, выполняющих базовые операции с базовыми типами и структурами данных для решения несложных задач. Большая часть задач допускает множество разнообразных решений, отличающихся по сложности реализации и эффективности, что позволяет выставлять дифференцированную оценку выполненным работам.

Для каждой лабораторной работы приведены краткие теоретические сведения, описание некоторых деталей реализации, некоторые рекомендации по принятию решений в стандартных ситуациях. Кроме того, для каждой лабораторной работы сформулированы контрольные вопросы, которые могут быть использованы для проверки понимания основных понятий и свойств. В конце каждой главы приведён пример выполнения одного из вариантов лабораторной работы на языке C++, что позволяет лучше разобраться в требованиях и принципах выполнения лабораторной работы.

Основной целью настоящих лабораторных работ является обучение основам программирования, общим для любого языка программирования и для любой прикладной задачи. Приобретённые за счёт выполнения этих лабораторных работ умения и навыки могут пригодиться на практике для любой деятельности, связанной с информационными технологиями. Возможность выбора между выполнением работы лёгким, но неэффективным способом и сложным, но более эффективным моделирует точно такие же реальные ситуации, возникающие при написании реальных приложений.

Информационные технологии и связанная с ними индустрия стремительно развиваются уже на протяжении полувека, но интен-

сивность их развития нисколько не снижается. Это влечёт потребность в большом количестве специалистов в этих областях, способных самостоятельно решать большинство практических задач. Хотя информатика стала крайне широкой областью знаний, и квалифицированные работники в этой сфере обычно специализируются на чём-то одном, им всем необходимы некоторые базовые знания для понимания основных принципов функционирования вычислительных устройств. И эти знания невозможно получить без реальной практики написания простейших приложений, таких как предложенные в этом издании.

Список литературы

1. Александреску, А. Современное проектирование на C++: Обобщенное программирование и прикладные шаблоны проектирования / А. Александреску. – СПб.: Вильямс, 2008. – 336 с.

2. Вирт, Н. Алгоритмы + структуры данных = программы / Н. Вирт. – М.: Мир, 1985. – 406 с.

3. Дасгупта, С. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани. – М.: МЦНМО, 2014. – 320 с.

4. Каймин, В. А. Информатика: Учебник / В. А. Каймин. – М.: ИНФРА-М, 2001. – 272 с.

5. Кнут, Д. Искусство программирования, том 1. Основные алгоритмы / Д. Кнут. – М.: Вильямс, 2006. – 720 с.

6. Кнут, Д. Искусство программирования, том 2. Получисленные алгоритмы / Д. Кнут. – М.: Вильямс, 2007. – 832 с.

7. Кнут, Д. Искусство программирования, том 3. Сортировка и поиск / Д. Кнут. – М.: Вильямс, 2007. – 824 с.

8. Кормен, Т. Х. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. – М.: Вильямс, 2005. – 1296 с.

9. Королев, Л. Информатика. Введение в компьютерные науки / Л. Королев, А. И. Миков. – М.: Высшая школа, 2003. – 342 с.

10. Липпман, С. Б. Основы программирования на C++ / С. Б. Липпман. – М.: Вильямс, 2002. – 256 с.

11. Макарова, Н. В. Информатика: Учебник для вузов / Н. В. Макарова, В. Б. Волков. – СПб.: Питер, 2011. – 576 с.

12. Майерс, С. Эффективное использование C++. 35 новых способов улучшить стиль программирования / С. Майерс. – СПб.: Питер, 2006. – 224 с.

13. Майерс, С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов / С. Майерс. – СПб.: Питер, 2006. – 240 с.

14. Окулов, С. М. Основы программирования / С. М. Окулов. – М.: БИНОМ. Лаборатория знаний, 2004. – 424 с.

15. Окулов, С. М. Программирование в алгоритмах / С. М. Окулов. – М.: БИНОМ. Лаборатория знаний, 2004. – 341 с.

16. Павловская, Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб.: Питер, 2003. – 461 с.

17. Савельев, А. Я. Основы информатики: Учеб. для вузов / А. Я. Савельев. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2001. – 328 с.

18. Саттер, Г. Новые сложные задачи на C++ / Г. Саттер. – М.: Вильямс, 2015. – 272 с.

19. Саттер, Г. Решение сложных задач на C++ / Г. Саттер. – М.: Вильямс, 2015. – 400 с.

20. Саттер, Г. Стандарты программирования на C++ / Г. Саттер, А. Александреску. – М.: Вильямс, 2015. – 224 с.

21. Седжвик, Р. Алгоритмы на C++. Фундаментальные алгоритмы и структуры данных / Р. Седжвик. – М.: Вильямс, 2011. – 1056 с.

22. Страуструп, Б. Дизайн и эволюция C++ / Б. Страуструп. – М.: ДМК Пресс, 2014. – 446 с.

23. Страуструп, Б. Язык программирования C++ / Б. Страуструп. – СПб.: Невский Диалект, 2001. – 1099 с.

24. Халперн, П. Стандартная библиотека C++ на примерах / П. Халперн. – М.: Вильямс, 2001. – 336 с.

25. Шень, А. Программирование: теоремы и задачи / А. Шень. – М.: МЦНМО, 2004. – 296 с.

26. Шилдт, Г. C++: базовый курс / Г. Шилдт. – М.: Вильямс, 2012. – 624 с.

27. Шилдт, Г. Теория и практика C++ / Г. Шилдт. – СПб.: BHV, 1996. – 416 с.

28. Шилдт, Г. Самоучитель C++ / Г. Шилдт. – СПб.: БХВ-Петербург, 2003. – 688 с.

29. Эккель, Б. Философия C++. Введение в стандартный C++ / Б. Эккель. – СПб.: Питер, 2004. – 572 с.

30. Эккель, Б. Философия C++. Практическое программирование / Б. Эккель. – СПб.: Питер, 2004. – 608 с.

31. Элджер, Дж. C++: Библиотека программиста / Дж. Элджер. – СПб.: Питер, 1999. – 320 с.

Учебное издание

Андрей Викторович Гайдель,
Александр Григорьевич Храмов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО КУРСУ «ОСНОВЫ ИНФОРМАТИКИ»

Практикум

Редактор И.И. Спиридонова
Компьютерная верстка И.И. Спиридоновой

Подписано в печать 22.08.2019. Формат 60×84 1/16.
Бумага офсетная. Печ. л. 10,75.
Тираж 25 экз. Заказ . Арт. – 13(Р2П)/2019.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

Изд-во Самарского университета.
443086, Самара, Московское шоссе, 34.