Министерство образования и науки, молодежи и спорта Украины
Донбасская государственная машиностроительная академия
Кафедра Прикладной математики

С. Л. Загребельный

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО ПРОГРАММИРОВАНИЮ В СРЕДЕ MICROSOFT VISUAL STUDIO 2010

по дисциплине

«Вычислительная техника и программирование»

Для студентов специальности 8.05070204 «Электромеханические системы автоматизации и электропривод»

Краматорск 2011

ВВЕДЕНИЕ

Данное учебное пособие освещает практические приемы программирования на языке С (читается "Си") в среде программирования Microsoft Visual Studio 2010, которая устанавливается в режиме программирования С.

Изначально язык С предназначался для системного программирования при создании операционных систем, системных утилит и встраиваемого программного обеспечения. Он обладает всеми необходимыми для этого свойствами: программы, написанные на нем, очень эффективны, не требуют специальной среды поддержки времени выполнения. Программы на языке С имеют низкие требования к аппаратной части вычислительной системы. Тем не менее в настоящее время язык С часто выбирается из-за стабильности языка и его окружения (стандартные библиотеки, компиляторы и другие инструментальные средства), а также наличия возможности получения программ, выполняющихся с максимальной скоростью на данной аппаратной платформе. Более того, язык С можно использовать и для создания веб-сайтов через технологию CGI (Common Gateway Interface – общий шлюзовый интерфейс).

Немаловажно также то, что компиляторы, библиотеки и инструменты разработки на языке С существуют практически для всех систем. Программы на языке С отличаются переносимостью между платформами на уровне исходного кода.

Язык С оказал большое влияние на индустрию разработки программного обеспечения. С одной стороны, синтаксис многих его инструкций лежит в основе таких языков, как C++, C#, Java, PHP. С другой – он используется в качестве промежуточного в некоторых системах программирования, когда программа сначала транслируется в программу на языке С, и только потом компилируется компилятором языка С для получения окончательного исполняемого модуля.

Язык С называют компьютерным языком "среднего уровня". Но это не означает, что он менее совершенен по сравнению с традиционными языками высокого уровня, такими как Fortran, Pascal, Basic и др. Язык С сочетает элементы языков высокого уровня с функциональностью ассемблера. В нем заложены возможности для разработки конструкций, характерных для языков высокого уровня. В то же время С позволяет манипулировать битами, байтами и адресами, т. е. базовыми элементами, с которыми работает компьютер.

К неоспоримым достоинствам языка С относятся следующие:

- универсальность (используется почти на всех существующих ЭВМ);
- компактность и универсальность кода;
- быстрота выполнения программ;
- гибкость;
- высокая структурированность.

Строительными блоками языка С являются функции, с помощью которых возможно выполнение операций как высокого, так и сравнительно низкого уровня.

Важным аспектом языка С является его структурированность. Специфическая черта структурированного языка – использование блоков. Блок – это набор инструкций, которые логически связаны между собой.

Другая характерная особенность языка С – отсутствие ответственности за действия программиста. Например, в нем не предусматривается контроль выхода за границы массивов (числовых или символьных). Основной принцип данного языка состоит в том, чтобы позволить программисту делать все, что он хочет, но и за последствия отвечает не язык, а программист.

ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ И ОФОРМЛЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1. **Номер варианта** лабораторной работы выбирается в соответствии с порядковым номером студента в журнале группы.

2. Каждый студент создает на своем сетевом диске **N** папку **CI** (должна быть набрана английскими буквами), в которую помещаются все результаты работы. Результат каждой лабораторной работы сохранять в отдельной папке (**Lab_1, lab_2, lab_3** и т.д.). Название проектов набирать английскими буквами.

3. Работа предъявляется на экране компьютера и в виде отчетов, содержащих: ФИО студента, шифр группы, номер и тему лабораторной работы, полный код программы на языке программирования СИ++. Отчет выполняется на компьютере и распечатывается на листах бумаги формата А4. Каждая работа должна быть защищена.

4. Перед каждой лабораторной работой студент должен самостоятельно проработать теоретический материал, относящийся к теме работы

Все проекты должны быть сохранены!!!

В конце семестра все проекты по всему курсу предъявляются преподавателю, принимающему экзамен или ведущему лабораторные работы.

ЛАБОРАТОРНАЯ РАБОТА 1

Знакомство со средой Microsoft Visual Studio 2010 и настройка компилятора языка C++. Стандартный ввод-вывод. Создание простейшей программы на языке C++.

Теоретическая часть

Язык **C** (читается как Си) в основе своей был создан в 1972 г. как язык для операционной системы UNIX . Автором этого языка считается Денис М. Ритчи (DENNIS M. RITCHIE).

Популярность языка **C** обусловлена, прежде всего тем, что большинство операционных систем были написаны на языке C. Его начальное распространение было задержано из-за того, что не было удачных компиляторов.

Несколько лет не было единой политики в стандартизации языка **C**. В начале 1980-х г. в Американском национальном институте стандартов (ANSI) началась работа по стандартизации языка **C**. В 1989 г. работа комитета по языку C была ратифицирована, и в 1990 г. был издан первый официальный документ по стандарту языка **C**. Появился стандарт 1989.

К разработке стандарта по языку **C** была также привлечена Международная организация по стандартизации (ISO). Появился стандарт ISO/IEC 9899:1990, или ANSI C99 языка **C**.

В данном пособии за основу принимается стандарт языка **C** от 1989 г. и написание программ будет выполняться в среде разработки Visual Studio 2010.

Язык **C** является прежде всего языком высокого уровня, но в нем заложены возможности, которые позволяют программисту (пользователю) работать непосредственно с аппаратными средствами компьютера и общаться с ним на достаточно низком уровне. Многие операции, выполняемые на языке **C**, сродни языку Ассемблера. Поэтому язык C часто называют языком среднего уровня.

Для написания программ в практических разделах данного учебного пособия будет использоваться компилятор языка **C++**, а программирование будет вестись в среде **Microsoft Visual Studio 2010**. Предполагается, что на компьютере установлена эта интегрированная среда.

Microsoft Visual Studio 2010 доступна в следующих вариантах:

- **Express** – бесплатная среда разработки, включающая только базовый набор возможностей и библиотек.
- **Professional** – поставка, ориентированная на профессиональное создание программного обеспечения, и командную разработку, при которой созданием программы одновременно занимаются несколько человек.
- **Premium** – издание, включающее дополнительные инструменты для работы и исходным кодом программ и создания баз данных.

- **Ultimate** – наиболее полное издание Visual Studio, включающие все доступные инструменты для написания, тестирования, отладки и анализа программ, а также дополнительные инструменты для работы с базами данных и проектирования архитектуры ПО.

Отличительной особенностью среды **Microsoft Visual Studio 2010** является то, что она поддерживает работу с несколькими языками программирования и программными платформами. Поэтому, перед тем, как начать создание программы на языке С, необходимо выполнить несколько подготовительных шагов по созданию проекта и выбора и настройки компилятора языка С для трансляции исходного кода

После запуска **Microsoft Visual Studio 2010** появляется следующая стартовая страница, которая показана на рис. 1.1.

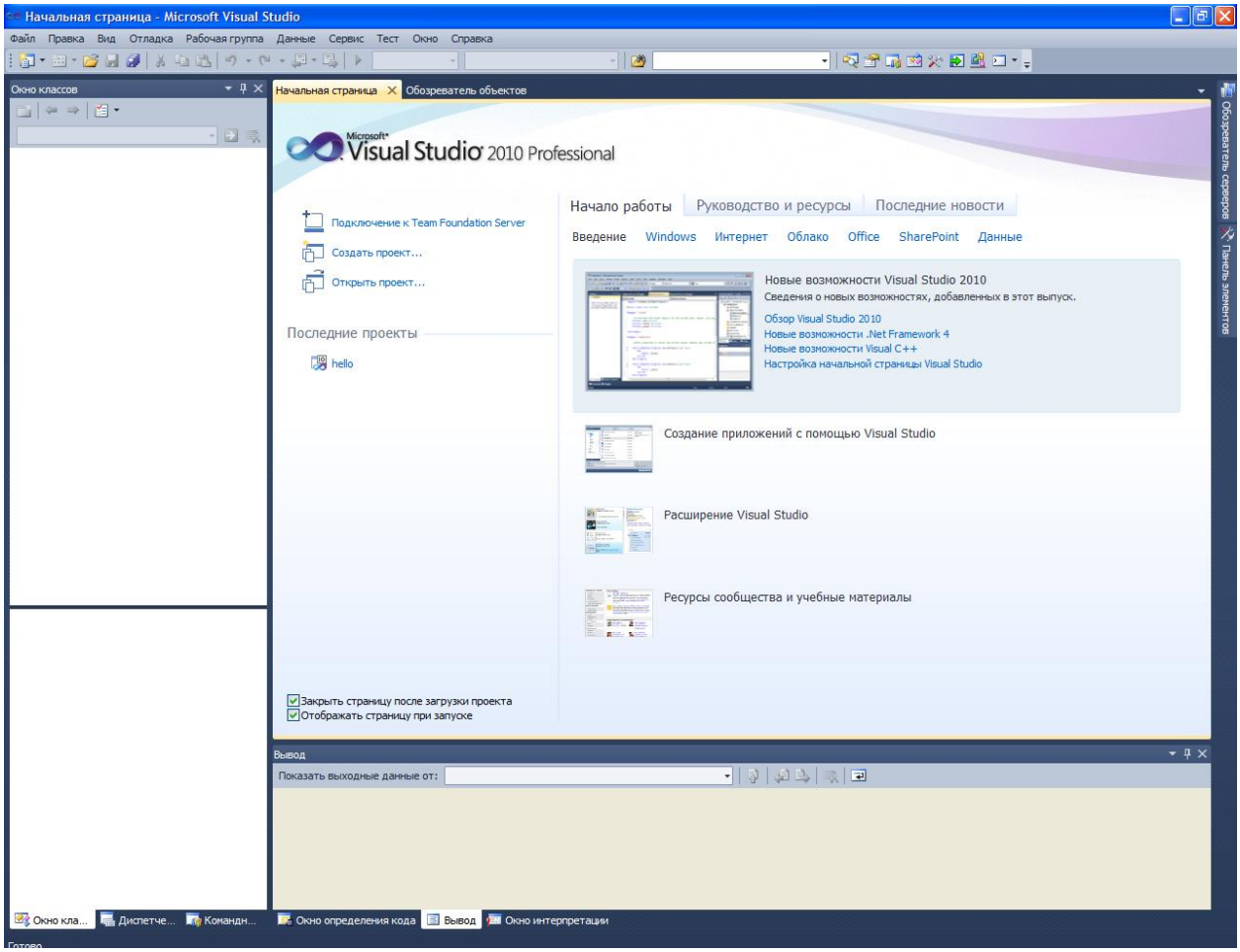


Рис. 1.1. Стартовая страница Visual Studio 2010

Следующим шагом является создание нового проекта. Для этого в меню **Файл** необходимо выбрать **Создать →Проект** (или комбинацию клавиш **Ctrl + Shift + N**). Результат выбора пунктов меню для создания нового проекта показан на рис. 1.2.

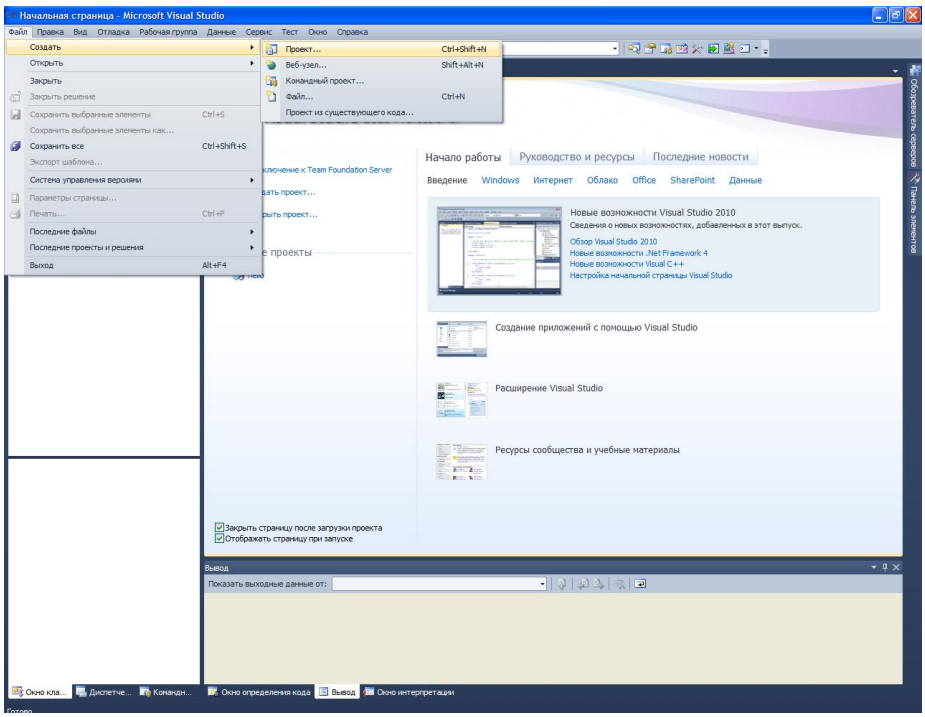


Рис. 1.2. Окно с выбором нового проекта

Среда Visual Studio отобразит окно **Создать Проект**, в котором необходимо выбрать тип создаваемого проекта. Проект используется в Visual Studio для логической группировки нескольких файлов, содержащих исходный код, на одном из поддерживаемых языков программирования, а также любых вспомогательных файлов. Обычно после сборки проекта (которая включает компиляцию всех всходящих в проект файлов исходного кода) создается один исполняемый модуль.

В окне **Создать Проект** следует развернуть узел Visual C++, обратиться к пункту Win32 и на центральной панели выбрать *Консольное приложение Win32*. Выбор этой опции показан на рис. рис. 1.3.

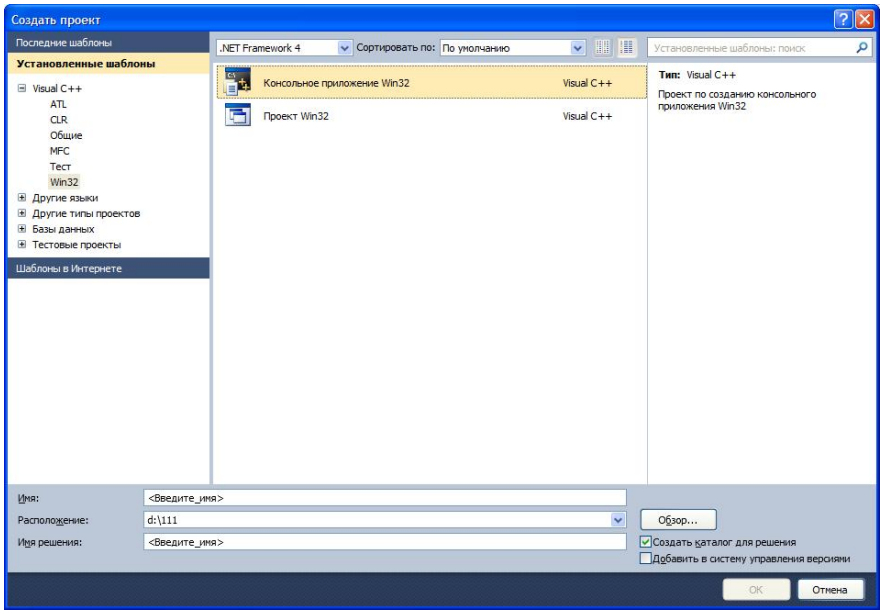


Рис. 1.3. Выбор типа проекта

Затем в поле редактора **Имя** (где по умолчанию имеется <Введите имя>) следует ввести имя проекта, например, **hell**. В поле **Расположение** можно указать путь размещения проекта, или выбрать путь размещения проекта с помощью клавиши (кнопки) Обзор. По умолчанию проект сохраняется в специальной папке Projects. Пример выбора имени проекта показано на рис. 1.4.

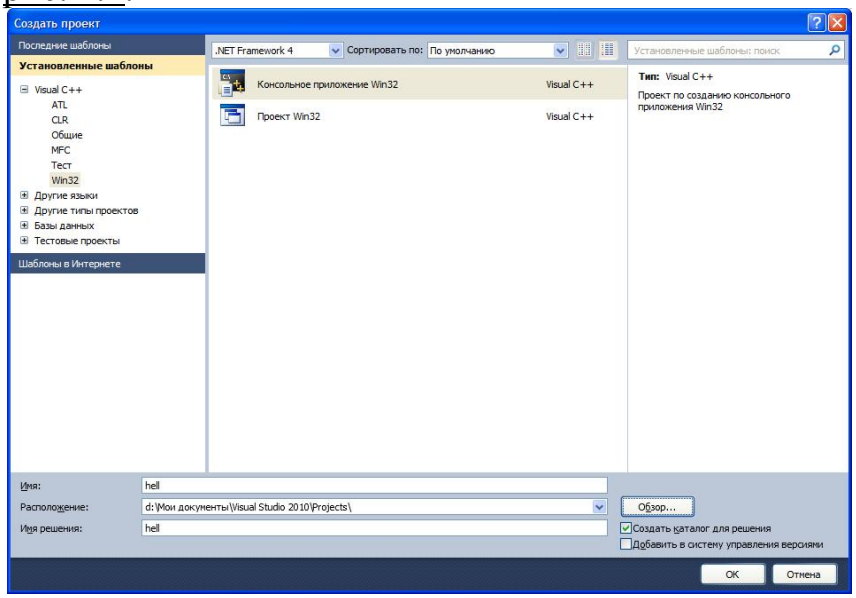


Рис. 1.4. Пример задания имени проекта

Одновременно с созданием проекта Visual Studio создает решение. Решение (solution) – это способ объединения нескольких проектов для организации более удобной работы с ними.

После нажатия кнопки **OK** откроется окно **Мастер приложений Win32**, показанное на рис. 1.5.

Выбор имени проекта может быть достаточно произвольным: допустимо использовать числовое значение, допустимо имя задавать через буквы русского алфавита.

В дальнейшем будем использовать имя, набранное с помощью букв латинского алфавита и, может быть, с добавлением цифр.

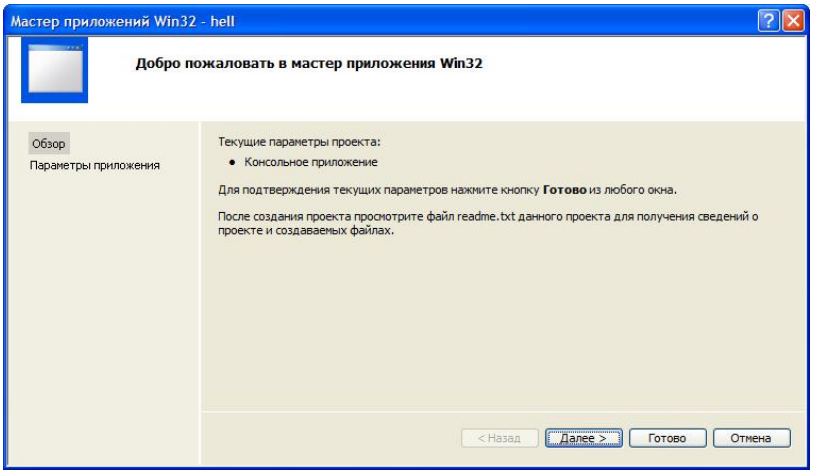


Рис. 1.5. Мастер создания приложения

На первой странице представлена информация о создаваемом проекте, на второй можно сделать первичные настройки проекта. После обращения к странице **Параметры приложения**, или после нажатия кнопки **Далее** получим окно, показанное на рис. рис. 1.6.

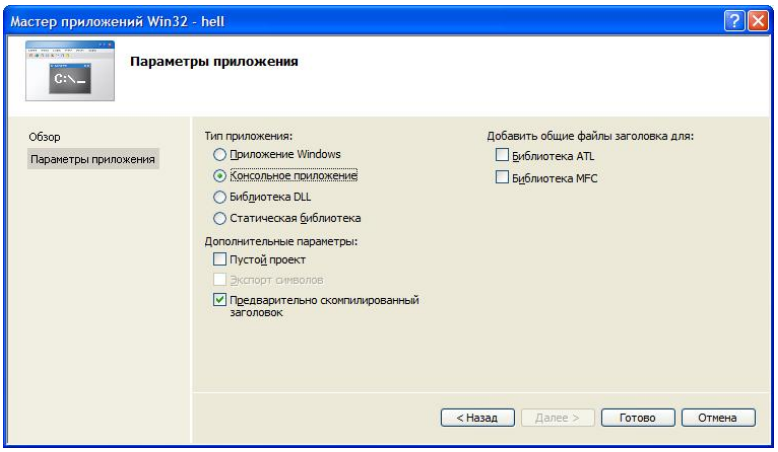


Рис. 1.6. Страница мастера настройки проекта по умолчанию

В дополнительных опциях (**Параметры приложения**) следует поставить галочку в поле **Пустой проект** и убрать галочку в поле **Предварительно скомпилированный заголовок**. Получим экранную форму, показанную на рис. 1.7.

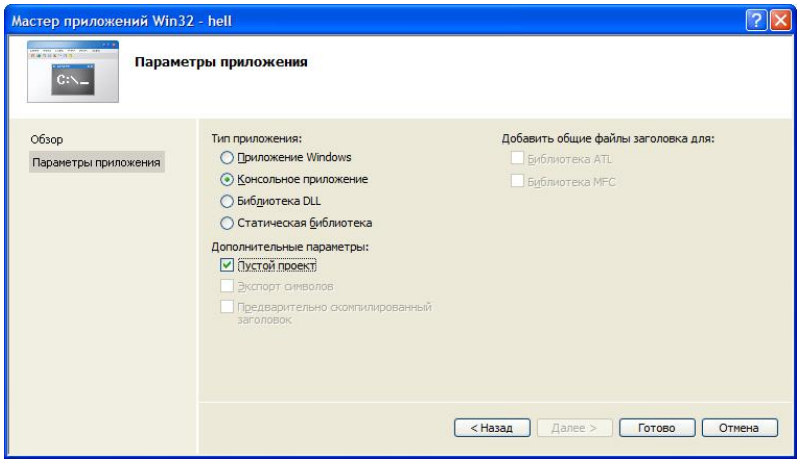


Рис. 1.7. Выполненная настройка мастера приложений

Здесь и далее будут создавать проекты по приведенной схеме, т.е. проекты в консольном приложении, которые должны создаваться целиком программистом (за счет выбора **Пустой проект**). После нажатия кнопки **Готово**, получим экранную форму, показанную на рис. 1.8, где приведена последовательность действий добавления файла для создания исходного кода к проекту. Стандартный путь для этого: подвести курсор мыши к пункту **Проект**, выбрать **Добавить новый элемент**.

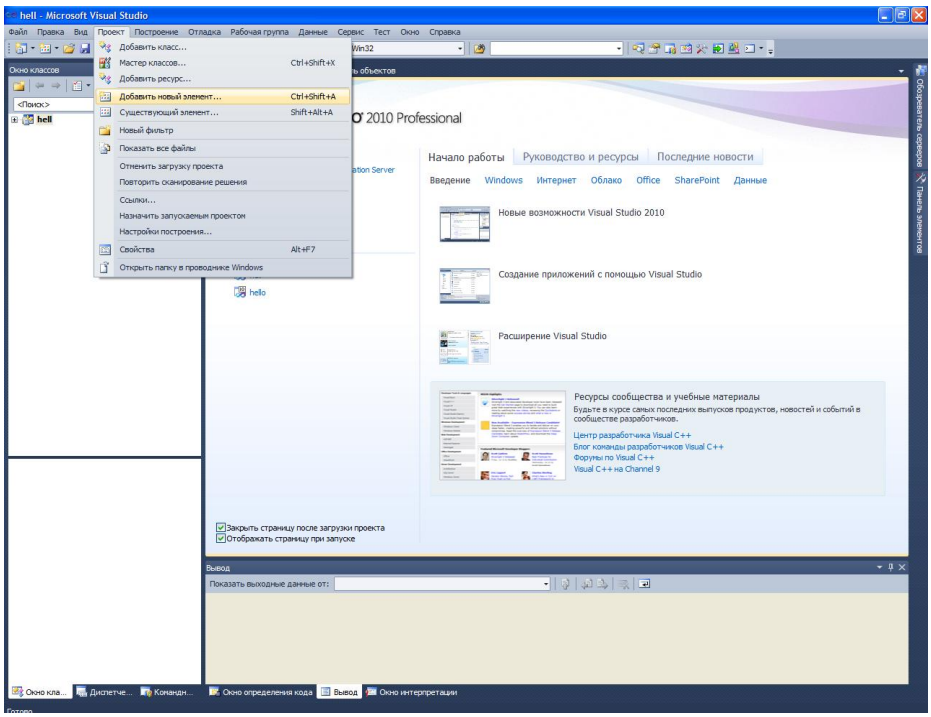


Рис. 1.8. Меню добавления нового элемента к проекту

После выбора (нажатия) **Новый элемент** получим окно, показанное на рис. 1.9, где через пункт меню **Код** узла **Visual C++** выполнено обращение к центральной части панели, в которой осуществляется выбор типа файлов. В данном случае требуется обратиться к закладке **C++ File** (.cpp).

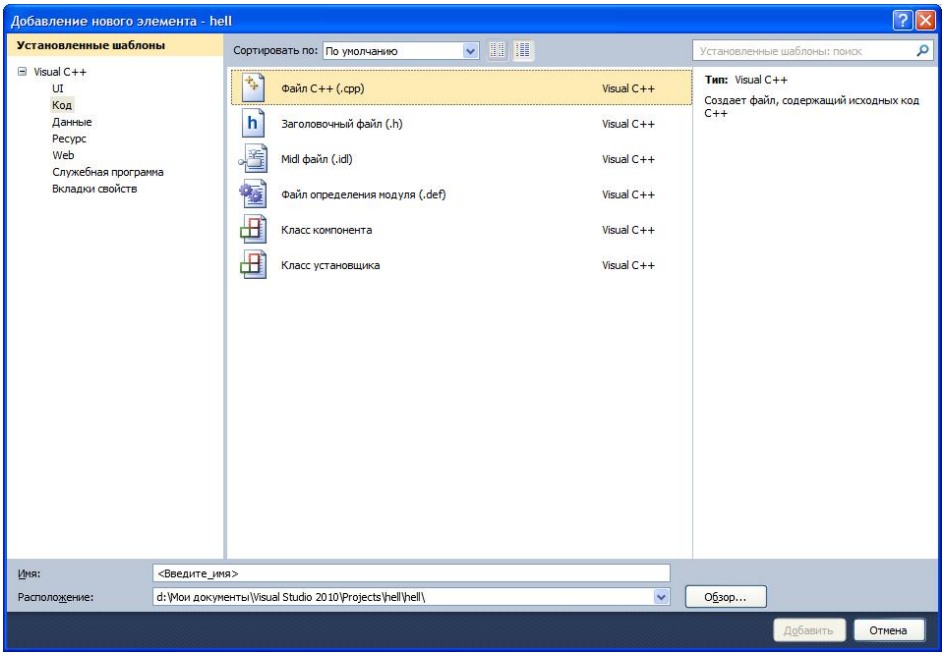


Рис. 1.9. Окно выбора типа файла для подключения к проекту

Теперь в поле редактора **Имя** (в нижней части окна) следует задать имя нового файла и указать расширение **".c"**. Например, **main.c**. Имя файла может быть достаточно произвольным, но имеется негласное соглашение, что имя файла должно отражать его назначение и логически описывать

исходный код, который в нем содержится. В проекте, состоящем из нескольких файлов, имеет смысл выделить файл, содержащий главную функцию программы, с которой она начнет выполняться. В данном пособии такому файлу мы будем задавать имя **main.c**, где расширение **.c** указывает на то, что этот файл содержит исходный код на языке **C**, и он будет транслироваться соответствующим компилятором. Программам на языке **C** принято давать расширение **.c**. После задания имени файла в поле редактора **Name**, получим форму, показанную на рис. 1.10.

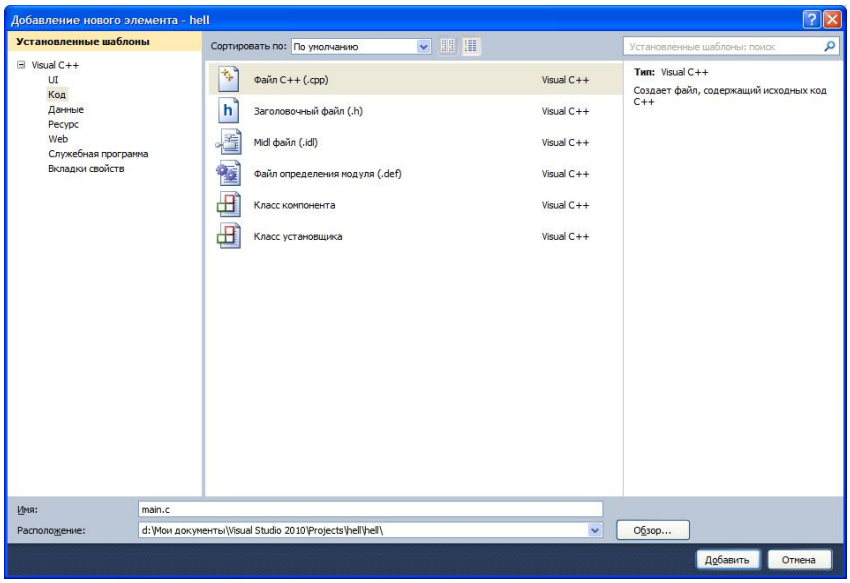


Рис. 1.10. Задание имени файла, подключаемому к проекту

Затем следует нажать кнопку **Добавить**. Вид среды Visual Studio после добавления первого файла к проекту показан на рис. 1.11. Для добавленного файла автоматически открывается редактор.

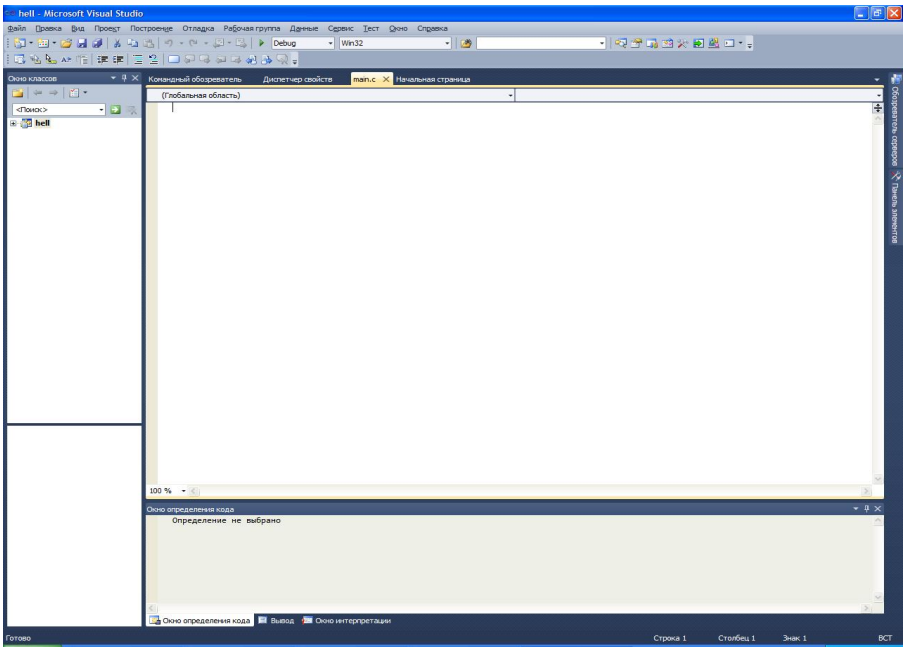


Рис. 1.11. Подключение файла проекта

В папке проекта отображаются файлы, включенные в проект в папках. Приведем описание.

Папка **Файлы исходного кода** предназначена для файлов с исходным кодом. В этой папке отображаются файлы с расширением **.c**.

Папка **Заголовочные файлы** содержит заголовочные файлы с расширением **.h.**

Папка **Файлы ресурсов** содержит файлы ресурсов, например изображения и т. д.

Папка **Внешние зависимости** отображает файлы, не добавленные явно в проект, но использующиеся в файлах исходного кода, например включенные при помощи директивы #include. Обычно в папке **Внешние зависимости** присутствуют заголовочные файлы стандартной библиотеки, использующиеся в проекте.

Следующий шаг состоит в настройке проекта. Для этого в меню **Проект** главного меню следует выбрать **Свойства hell** (или с помощью последовательного нажатия клавиш Alt+F7). Пример обращения к этому пункту меню показан на рис. 1.12.

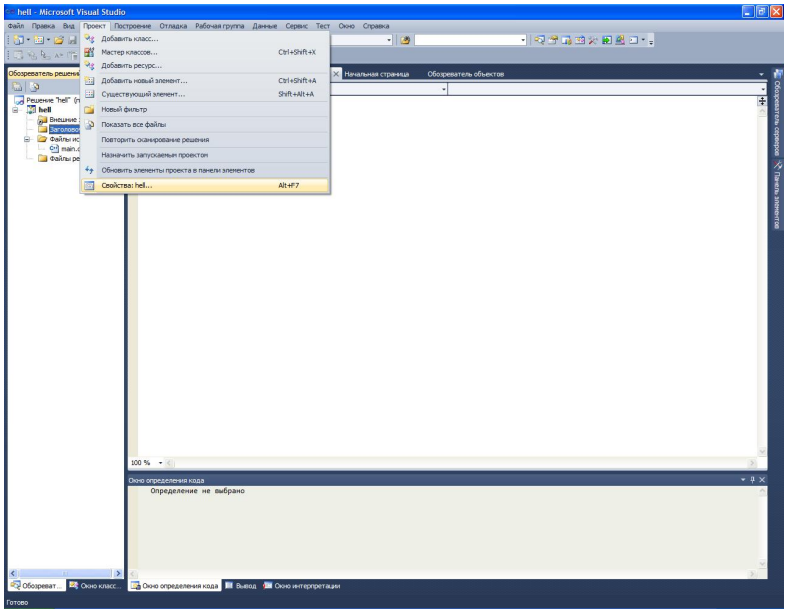


Рис. 1.12. Обращение к странице свойств проекта

После того как произойдет открытие окна свойств проекта, следует обратиться (с левой стороны) к **Свойства конфигурации**. Появится ниспадающий список, который показан на рис. 1.13. Выполнить обращение к узлу **Общие**, и через него в левой панели выбрать **Набор символов**, где установить свойство **Использовать многобайтовую кодировку**. Настройка **Набор символов** позволяет выбрать, какая кодировка символов – ANSI или UNICODE – будет использована при компиляции программы. Для совместимости со стандартом C89 мы выбираем **Использовать многобайтовую кодировку**. Это позволяет использовать многие привычные функции, например, функции по выводу информации на консоль.

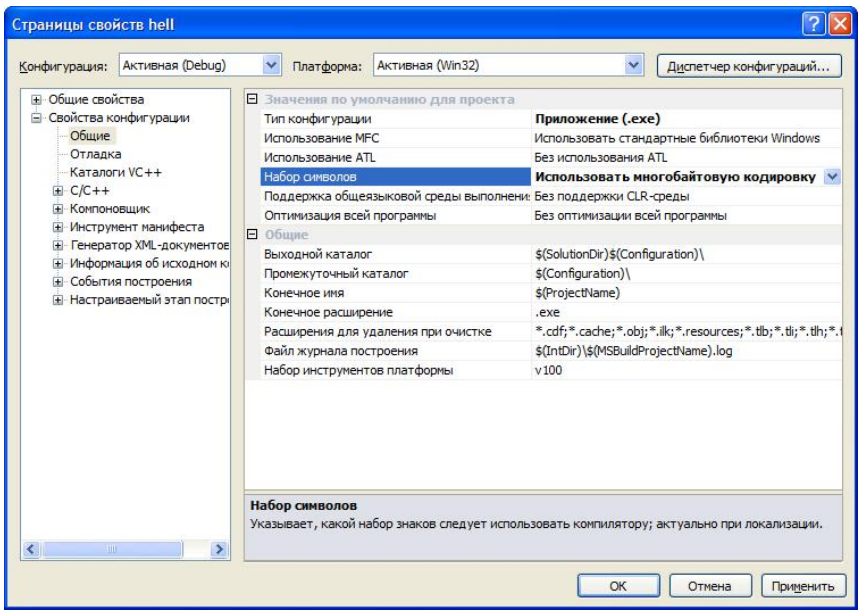


Рис. 1.13. Меню списка свойств проекта

После сделанного выбора, показанного на рис. 1.13, следует нажать кнопку Применить. Затем следует выбрать узел **C/C++** и в ниспадающем меню выбрать пункт **Создание кода**, через который следует обратиться в правой части панели к закладке **Включить C++ исключения**, для которой установить **Нет** (запрещение исключений C++). Результат установки выбранного свойства показан на рис. 1.14. После произведенного выбора нажать кнопку **Применить**.

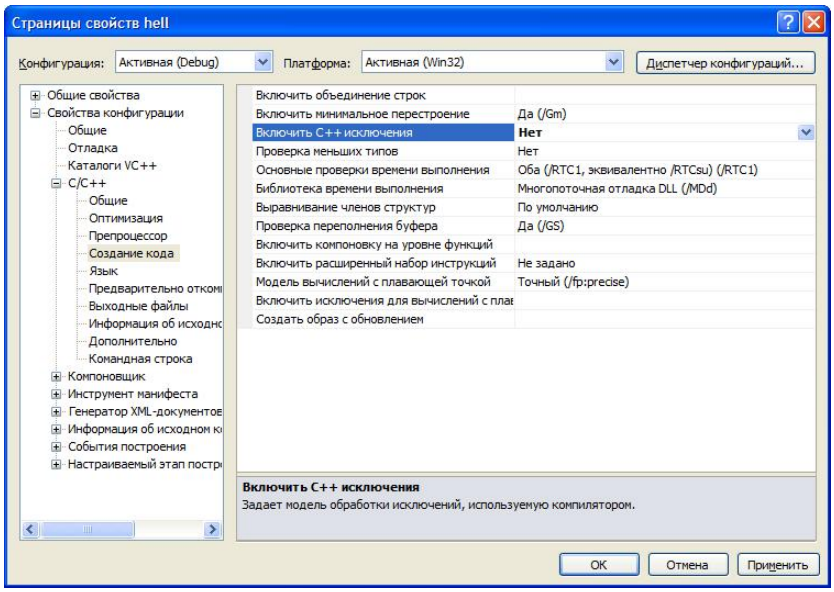


Рис. 1.14. Страница свойств для запрещения исключений C++

Далее в ниспадающем меню узла **C/C++** необходимо выбрать пункт **Язык** и через него обратиться в правую часть панели, где установить следующие свойства: свойство **Отключить расширение языка** (дополнительные языковые расширения фирмы Microsoft) в **Да (/Za)**, свойство **Считать wchar_t встроенным типом** установить в

Нет (/Zc:wchar_t–), свойство **Обеспечение согласования видимости переменных, объявленных в заголовке оператора цикла for** установить в да(/Zc:forScope), свойство **Включить информацию о типах время выполнения** установить в **Нет (/GR–)**, свойство **Поддержка Open MP** (разрешить расширение Open MP – используется при написании программ для многопроцессорных систем) установить в **Нет(/openmp–)**.

Результат выполнения этих действий показан на рис. 1.15.

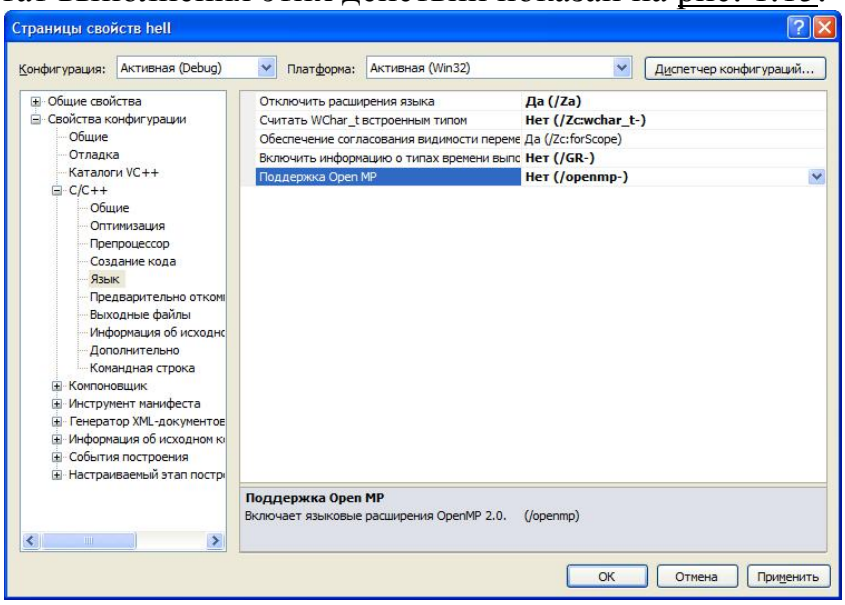


Рис. 1.15. Страница свойств закладки *Язык*

После выполнения указанных действий следует нажать клавишу **Применить**. Далее в ниспадающем списке узла **C/C++** следует выбрать пункт **Дополнительно** и в правой панели изменить свойство **Компилировать как** в свойство компиляции языка **C**, т.е. **Компилировать как код C (/TC)**. Результат установки компилятора языка **C** показан на рис. 1.16.

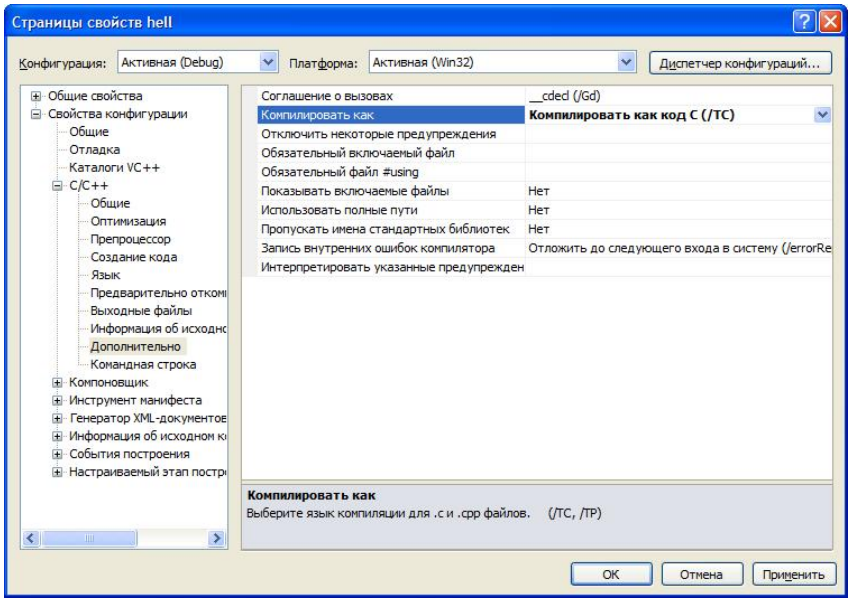


Рис. 1.16. Результат выбора режима компиляции языка C

После нажатия клавиш **Применить** и **ОК** сначала откроется подготовленный проект с пустым полем редактора кода, в котором можно начать писать программы. В этом редакторе наберем программу, выводящую традиционное приветствие "Hell World". Для компиляции созданной программы можно обратиться в меню **Построение**, или, например, набрать клавиши Ctr+F7. В случае успешной компиляции получим следующую экранную форму, показанную на рис. 1.17.

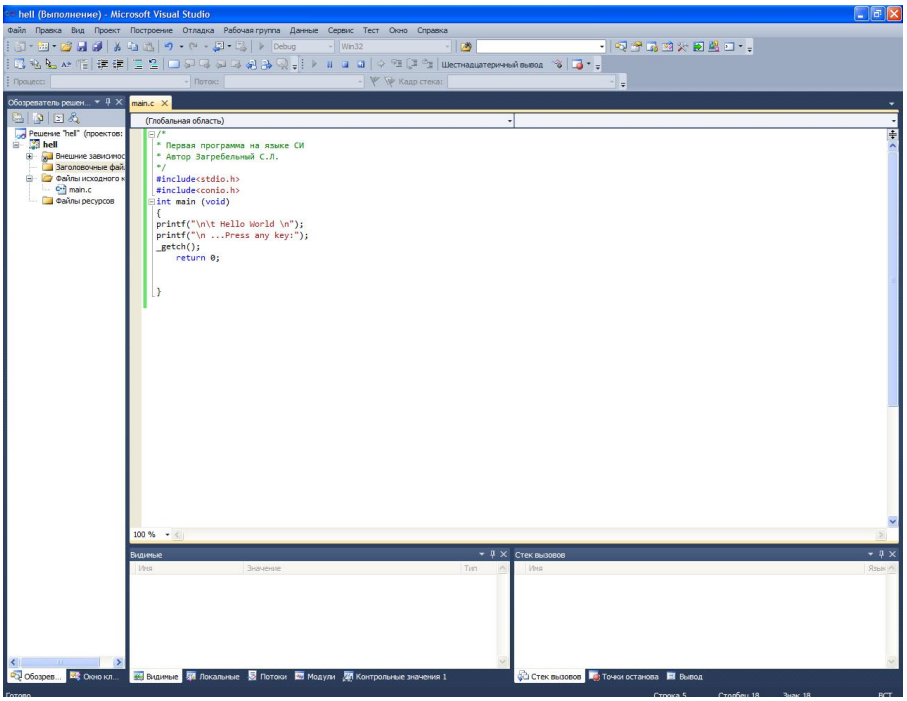


Рис. 1.17. Успешно откомпилированная первая программа на языке С

Для приведенного кода программы запуск на ее исполнение из окна редактора в Visual Studio 2010 можно нажать клавишу **F5**. рис. 1.18 показан результат исполнения первой программы.

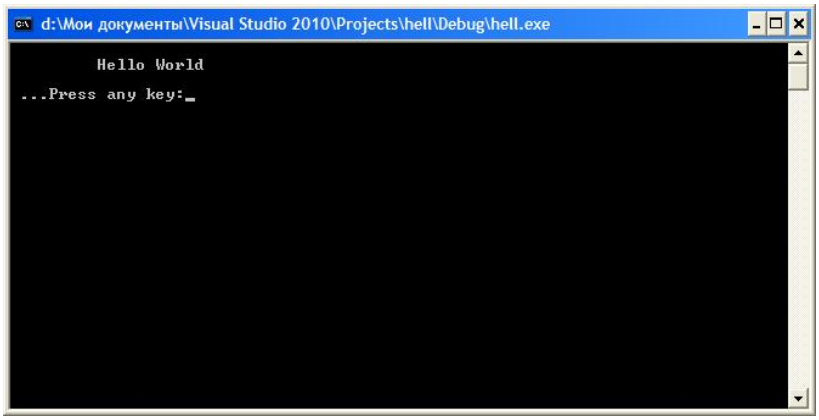


Рис. 1.18. Консольный вывод первой программы на языке С

Примечание. Вывод требуемой информации осуществляется с помощью букв латинского алфавита. Комментарии в программе могут быть

сделаны после символа "//" или внутри комбинации символов "/* */".

Произведем разбор первой программы. Во-первых, надо отметить, что в языке **C** нет стандартных инструкций (операторов) для вывода сообщений на консоль (окно пользователя). В языке **C** предусматриваются специальные библиотечные файлы, в которых имеются функции для этих целей. В приведенной программе используется заголовочный файл с именем stdio.h (стандартный ввод–вывод), который должен быть включен в начало программы. Для вывода сообщения на консоль используется функция printf(). Для работы с консолью включен также заголовочный файл conio.h., который поддерживает функцию _getch(), которая извлекает символ из потока ввода, т. е. она предназначенная для приема сообщения о нажатии какой-либо (почти любой) клавиши на клавиатуре. С другими компиляторами, возможно, потребуется getch(), т.е. без префиксного нижнего подчеркивания. Строка программы

```
int main (void)
```

сообщает системе, что именем программы является main() – главная функция, и что она возвращает целое число, о чем указывает аббревиатура **"int"**. Имя main() – это специальное имя, которое указывает, где программа должна начать выполнение [1.1]. Наличие круглых скобок после слова main() свидетельствует о том, что это имя функции. Если содержимое круглых скобок отсутствует или в них содержится служебное слово **void**, то это означает, что в функцию main() не передается никаких аргументов. Тело функции main() ограничено парой фигурных скобок. Все утверждения программы, заключенные в фигурные скобки, будут относиться к функции main().

В теле функции main() имеются еще три функции. Во-первых, функции printf() находятся в библиотеке компилятора языка **C**, и они печатают или отображают те аргументы, которые были подставлены вместо параметров. Символ "\n" составляет единый символ newline (новая строка), т.е. с помощью этого символа осуществляется перевод на новую строку. Символ "\t" осуществляет табуляцию, т.е. начало вывода результатов программы с отступом вправо.

Функция без параметров _getch() извлекает символ из потока ввода (т.е. ожидает нажатия почти любой клавиши). С другими компиляторами, возможно, потребуется getch(), т.е. без префиксного нижнего подчеркивания.

Последнее утверждение в первой программе

```
return 0;
```

указывает на то, что выполнение функции main() закончено и что в систему возвращается значение 0 (целое число). Нуль используется в соответствии с соглашением об индикации успешного завершения программы.

В завершение следует отметить, что все действия в программе завершаются символом точки с запятой.

Все файлы проекта сохраняются в той папке, которая сформировалась после указания в поле Location имени проекта (hell). На рис. 1.19 показаны

папки и файлы проекта Visual Studio 2010.

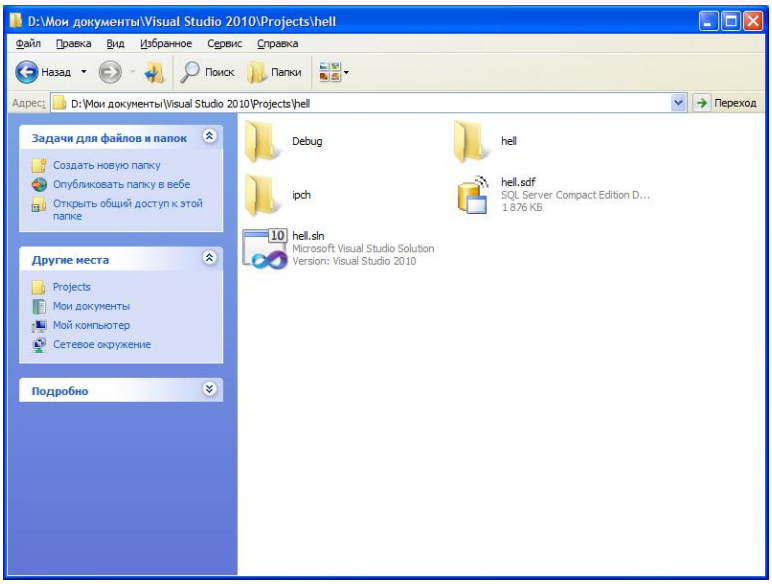


Рис. 1.19. Файлы и папки созданного проекта

На рис. 1.19 файлы с полученными расширениями означают:

hell.sln – файл решения для созданной программы. Он содержит информацию о том, какие проекты входят в данное решение. Обычно, эти проекты расположены в отдельных подкаталогах. Например, наш проект находится в подкаталоге hell;

hell.suo – файл настроек среды Visual Studio при работе с решением, включает информацию об открытых окнах, их расположении и прочих пользовательских параметрах.

hell.sdf – файл содержащий вспомогательную информацию о проекте, который используется инструментами анализа кода Visual Studio, такими как IntelliSense для отображения подсказок об именах и т.д.

Файлы папки **Debug** показаны на рис. 1.20.

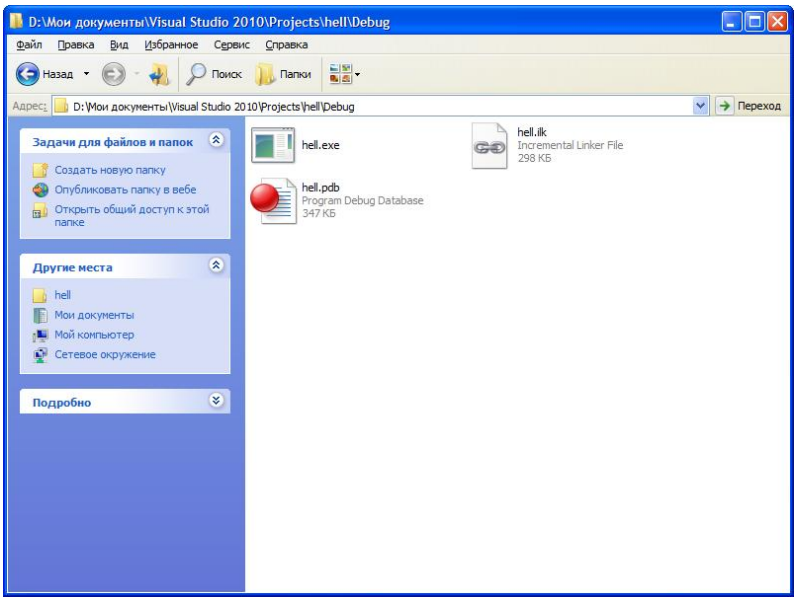


Рис. 1.20. Файлы папки Debug

Рассмотрим файлы в соответствии с рис. 1.20.

hell.exe – исполняемый файл проекта;

hell.ilk – файл "incremental linker", используемый компоновщиком для ускорения процесса компоновки;

hell.pdb – отладочная информация/информация об именах в исполняемых файлах, используемая отладчиком.

Файлы папки **hell** показаны на рис. 1.21.

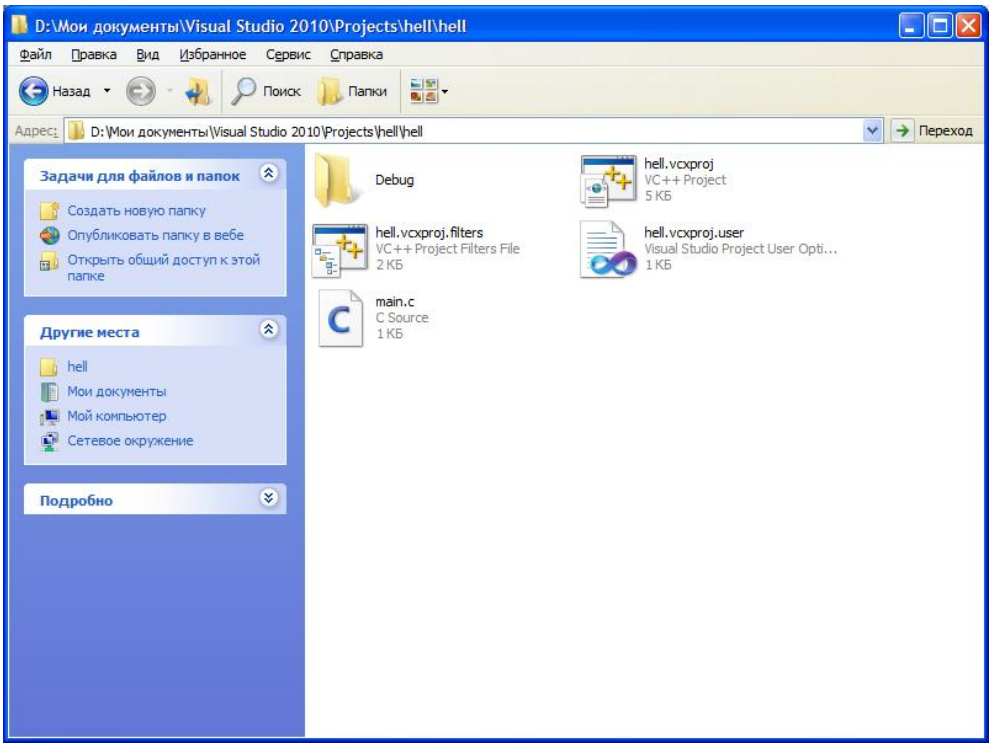


Рис. 1.21. Содержимое папки hell

Характеристика содержимого папки **hell**:

main.c – файл исходного программного кода,

hell.vcxproj – файл проекта,

hell.vcxproj.user – файл пользовательских настроек, связанных с проектом,

hell.vcxproj.filters – файл с описанием фильтров, используемых Visual Studio Solution Explorer для организации и отображения файлов с исходным кодом.

Задание к лабораторной работе

Разобраться с программой Visual Studio 2010 и суметь создать консольную программу по выше показанному примеру.

Контрольные вопросы

1. Какие компиляторы языка **C** вам известны?
2. Какое имя имеет исполняемый файл созданного проекта?
3. Объясните назначение заголовочных файлов stdio.h, conio.h.
4. Как будет работать программа без заголовочного файла conio.h?
5. В каком месте программы находится точка ее входа?
6. Как осуществляется табуляция строки на консоли и на сколько позиций выполняется отступ от левого края?
7. Какое значение имеет главная функция проекта main() в программах на языке **C**?

ЛАБОРАТОРНАЯ РАБОТА 2

Переменные и базовые типы данных языка С++. Создание программы линейного алгоритма

Теоретическая часть

2.1.1. Понятие алгоритма. Блок-схема

Алгоритм - конечная последовательность предписаний, однозначно определяющая процесс преобразования исходных данных в результат решения задачи.

В процессе разработки алгоритма могут использоваться различные способы его описания. Наиболее распространенные:

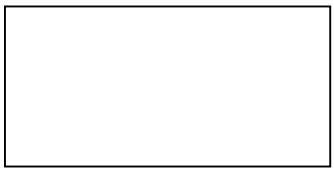
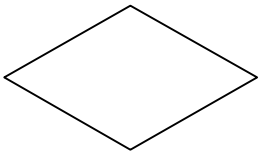
- словесная запись;
- графические схемы алгоритмов (блок-схемы);
- псевдокод (формальные алгоритмические языки);
- структурограммы.

Блок-схема - это графическое представление алгоритма, дополненное элементами словесной записи. На блок-схеме каждый пункт алгоритма изображается соответствующей геометрической фигурой. В таблице 2.1. приведены графические элементы, на которых компонуются блок-схемы, их названия и символы.

Таблица 2.1. - Графические элементы блок-схем

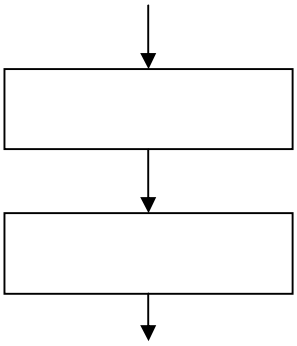
Название блока	**Блок**	**Отображаемая функция**
Начало-конец		Начало, конец, вход - выход в программах
Блок ввода-вывода		Ввод данных либо вывод результатов на экран
Блок вывода		Вывод данных на печать

Продолжение таблицы 2.1. - Графические элементы блок-схем

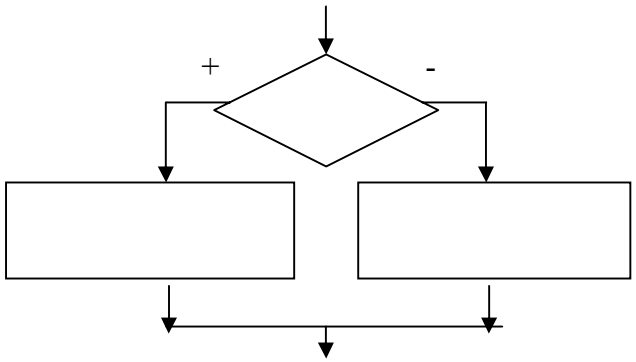
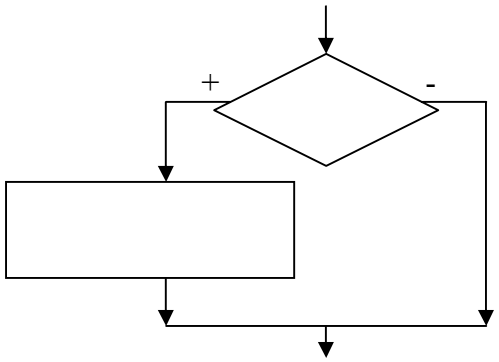
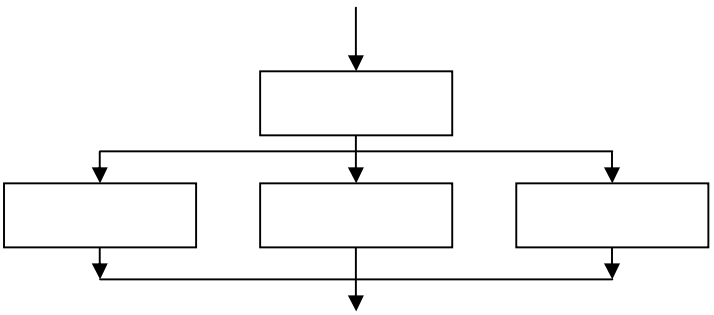
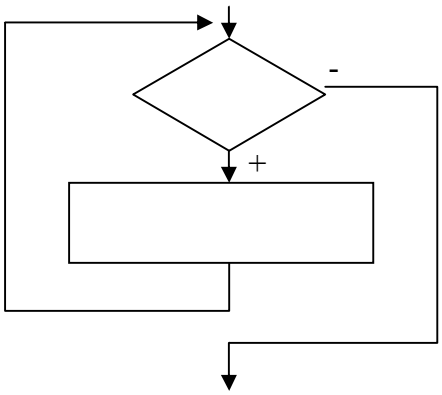
Процесс		Вычисление или последовательность вычислений
Предопределенный процесс		Выполнение подпрограммы
Альтернатива		Проверка условий
Модификация		Начало цикла
Соединитель		Разрыв линий потока информации в пределах одной страницы

В таблице 2.2. приведены основные базовые элементарные структуры для составления блок-схем.

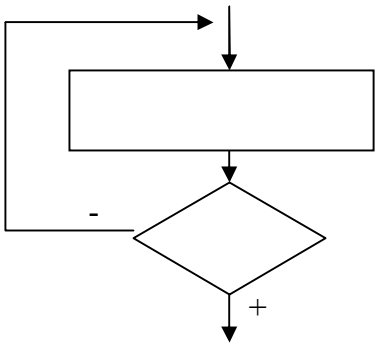
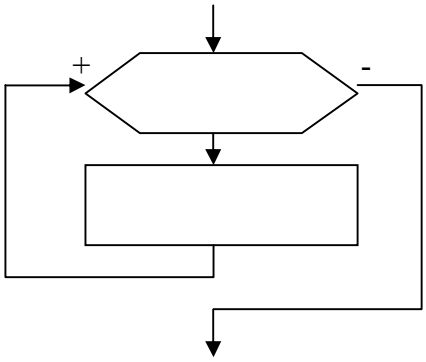
Таблица 2.2. - Базовые структуры блок-схем

Название типа структуры	**Изображение**
Последовательность	

Продолжение таблицы 2.2. – Базовые структуры блок-схем

Разветвление (выбор)	
Сокращенная запись разветвления	
Выбор варианта	
Цикл с предусловием	

Продолжение таблицы 2.2. – Базовые структуры блок-схем

Цикл с постусловием	
Цикл с параметрами	

2.1.2. Алфавит и лексемы языка Си++

В алфавит языка Си входят:

– прописные и строчные буквы латинского алфавита;
– цифры: 0,1,2,3,4,5,6,7,8,9;
– специальные знаки:

” {} , [] () | + - / \ % ;
‘ : < = > _ ! & # ^ . * ~

Из символов алфавита формируются лексемы языка:

– идентификаторы;
– ключевые (служебные, иначе зарезервированные) слова;
– константы;
– знаки операций;
– разделители (знаки пунктуации).

Рассмотрим эти лексические элементы языка подробнее.

Идентификатор - последовательность из букв латинского алфавита, десятичных цифр и символов подчеркивания, начинающаяся не с цифры:

```
RUN      run          hard_RAM_disk
copy_54
```

Прописные и строчные буквы различаются. Таким образом, в этом примере два первых идентификатора различны. На длину различаемой части идентификатора конкретные реализации накладывают ограничение.

Компиляторы различают не более 32-х первых символов любого идентификатора. Некоторые реализации Си++ на ЭВМ типа VAX допускают идентификаторы длиной до 8 символов.

Ключевые (служебные) слова - это идентификаторы, зарезервированные в языке для специального использования. Ключевые слова Си++:

```
asm             else            private         throw
auto            enum            protected       try
break           extern          public          typedef
case            float           register        typeid
catch           for             return          union
char            friend          short           unsigned
class           goto            signed          virtual
const           if              sizeof          void
continue        inline          static          volatile
default         int             struct          while
delete          long            switch
do              new             template
double          operator        this
```

Ранее в языке Си++ был зарезервирован в качестве ключевого слова идентификатор overload. Для компиляторов фирмы Borland (BC++ и TC++) дополнительно введены ключевые слова:

```
cdecl           _export         _loadds         _saveregs
_cs             far             near            _seg
_ds             huge            pascal          _ss
_es             interrupt       _regparam
```

Там же введены как служебные слова регистровые переменные:

```
_AH     _BL     _CX     _SI     _CS     _FLA
_AL     _BX     _DH     _DI     _DS     GS
_AX     _CH     _DL     _BP     _SS
_BH     _CL     _DX     _SP     _ES
```

Отметим, что ранние версии BC++ и TC++ не включали в качестве ключевых слов идентификаторы throw, try, typeid, catch.

Не все из перечисленных служебных слов сразу же необходимы программисту, однако их запрещено использовать в качестве произвольно выбираемых имен, и список служебных слов нужно иметь уже на начальном этапе знакомства с языком Си++. Кроме того, идентификаторы, включающие два подряд символа подчеркивания (__), резервируются для реализаций Си++ и стандартных библиотек. Идентификаторы, начинающиеся с символа подчеркивания (_), используются в реализациях языка Си. В связи с этим начинать выбираемые пользователем идентификаторы с символа

подчеркивания и использовать в них два подряд символа подчеркивания не рекомендуется.

Константа (литерал) - это лексема, представляющая изображение фиксированного числового, строкового или символьного (литерного) значения.

Константы делятся на пять групп: целые, вещественные (с плавающей точкой), перечислимые, символьные (литерные) и строковые (строки или литерные строки). Перечислимые константы проект стандарта языка Си++ [2] относит к одному из целочисленных типов.

Компилятор, выделив константу в качестве лексемы, относит её к той или другой группе, а внутри группы - к тому или иному типу данных по ее "внешнему виду" (по форме записи) в исходном тексте и по числовому значению.

Целые константы могут быть десятичными, восьмеричными и шестнадцатеричными.

Фундаментальные объекты данных, с которыми работает программа, – это переменные и константы. Используемые в программе переменные перечисляются в объявлениях или декларациях, в которых указывается их тип, а также иногда их начальные значения.

С именами переменных связывается тип данных, который контролируется компилятором и для которого выделяется определенное количество байтов памяти. Имена переменных должны начинаться с буквы (латинского алфавита) или символа подчеркивания (например, _aza), за которым могут следовать любые комбинации букв в любом регистре (заглавные или строчные), символы подчеркивания или цифры 0–9. В языке **C** имеется различие между заглавными и строчными буквами. Поэтому переменная World будет отличаться от переменной world и т.п. При этом в определении переменной не разрешается символ пробела (пробелов) и некоторые другие символы, например, $... .

Стандарт C89 определяет пять базовых типов данных:

int – целочисленный тип, целое число;

float – вещественное число одинарной точности с плавающей точкой;

double – вещественное число двойной точности с плавающей точкой;

char – символьный тип для определения одного символа;

void – тип без значения.

Кроме того, существуют модификаторы, которые могут применяться к этим базовым типам. Ряд компиляторов может поддерживать еще и логический тип _Bool. Тип void служит для объявления функции, не возвращающей значения, или для создания универсального указателя (pointer).

Объект типа char всегда занимает 1 байт памяти. Размеры объектов других типов, как правило, зависят от среды программирования и операционной системы.

Приведем модификаторы базовых типов данных. К ним относятся следующие спецификаторы, предшествующие им в тексте программы:

`signed, unsigned, long, short`

Базовый тип int может быть модифицирован каждым из перечисленных спецификаторов. Тип char модифицируется с помощью unsigned и signed, тип double – с помощью long.

В табл. 2.3 приведены допустимые комбинации типов данных языка **C** с их минимальным диапазоном значений и типичным размером.

Таблица 2.3.

Типы данных языка C

Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
char	8 (или 1 байт)	от –127 до 127
unsigned char	8	от 0 до 255
signed char	8	от –127 до 127
int	16 или 32	от –32767 до 32767
unsigned int	16 или 32	от 0 до 65535
signed int	16 или 32	от –32767 до 32767
short int	16	от –32767 до 32767
unsigned short int	16	от 0 до 65535
signed short int	16	от –32767 до 32767
long int	32	от –2147483647 до 2147483647
long long int	64	от $-(2^{63}-1)$ до $(2^{63}-1)$ для C99
signed long int	32	от –2147483647 до 2147483647
unsigned long int	32	от 0 до 4294967295
unsigned long long int	64	от 0 до $(2^{64}-1)$ для C99
float	32	от 1E–37 до 1E+37 (с точностью не менее 6 значащих десятичных цифр)
double	64	от 1E–37 до 1E+37 (с точностью не менее 10 значащих десятичных цифр)
long double	80	от 1E–37 до 1E+37 (с точностью не менее 10 значащих десятичных цифр)

Для базового типа int возможны следующие записи с модификатором:

```
signed  или  signed int
unsigned  или  unsigned int
long  или  long int
short  или  short int
```

Для данных вещественного типа максимальные значения абсолютных величин представлены в табл. 2.4 .

Таблица 2.4.

Вещественные типы данных языка С

Тип данных	Типичный размер в битах	Диапазон абсолютных величин
float	32	от 3.4E–38 до 3.4E+37
double	64	от 1.7E–308 до 1.7E+308
long double	80	от 3.4E–4932 до 1.1E+4932

В языке **С** предусматривается преобразование типов в выражениях и приведение типов. Если в выражении смешаны различные типы литералов и переменных, то компилятор преобразует их в один тип. Во-первых, все char и short int значения автоматически преобразуются (с расширением "типоразмера") в тип int. Этот процесс называется целочисленным расширением (integral promotion). Во-вторых, все операнды преобразуются (также с расширением "типоразмера") в тип самого большого операнда. Этот процесс называется расширением типа (type promotion), причем он выполняется пооперационно. Например, если один операнд имеет тип int, а другой–long int, то тип int расширяется в тип long int. Или если хотя бы один из операндов имеет тип double, то любой другой операнд приводится к типу double. Это означает, что такие преобразования, как тип char в тип double, вполне допустимы (если предусматривать, к чему это приведет). После преобразования оба операнда будут иметь один и тот же тип, а результат операции – тип, совпадающий с типом операндов. Приведем последовательность преобразования типов в выражениях по старшинству [2.4]:

```
ЕСЛИ  операнд имеет тип long double
ТО  второй операнд преобразуется в long double
ИНАЧЕ ЕСЛИ  операнд имеет тип double
ТО  второй операнд преобразуется в double
ИНАЧЕ ЕСЛИ  операнд имеет тип float
ТО  второй операнд преобразуется в float
ИНАЧЕ ЕСЛИ  операнд имеет тип unsigned long
ТО  второй операнд преобразуется в unsigned long
ИНАЧЕ ЕСЛИ  операнд имеет тип long
ТО  второй операнд преобразуется в long
ИНАЧЕ ЕСЛИ  операнд имеет тип unsigned int
ТО  второй операнд преобразуется в unsigned int
```

Кроме того, действует правило: если один из операндов имеет тип `long`, а второй– `unsigned int`, притом значение `unsigned int` не может быть представлено типом `long`, то оба операнда преобразуются в значение типа `unsigned long`.

В языке С предусматривается явное преобразование (приведение) типов. Общая форма оператора явного приведения типа: (тип) выражение.

В приведенной форме тип – это любой поддерживаемый тип данных.

Явное преобразование типа – это операция. Оператор приведения типа является унарным и имеет тот же приоритет, что и остальные унарные операторы.

В приводимых ниже программах используются такие средства ввода-вывода, как **`printf(), getchar(), gets(), scanf().`**

Приведем характеристику данных функций.

Прототип функции `printf()` имеет вид:

```
int printf(const char *format, ?);
```

Функция printf() записывает в стандартный поток stdout (стандартный выходной поток данных) значения аргументов из заданного списка аргументов в соответствии со строкой форматирования, адресуемой параметром format. Строка форматирования состоит из элементов двух типов. К элементам первого типа относятся символы, которые выводятся на экран. Элементы второго типа содержат спецификации формата, определяющий способ отображения аргументов. Спецификация формата начинается символом процента, за которым следует код формата. На спецификации формата могут воздействовать модификаторы, задающие ширину поля, точность и признак выравнивания по левому краю. Целое значение, расположенное между знаком % и командой форматирования, играет роль спецификации минимальной ширины поля. Наличие этого спецификатора приводит к тому, что результат будет заполнен пробелами или нулями, чтобы выводимое значение занимало поле, ширина которого не меньше заданной С I минимальной ширины. По умолчанию в качестве заполнителя используется пробел (пробелы). Для заполнения нулями перед спецификацией ширины поля нужно поместить нуль, т.е. 0. Например, спецификация формата %05d дополнит нулями выводимое целое число, в котором менее пяти цифр, чтобы общая длина равнялась пяти символам. Действие модификатора точности зависит от кода формата, к которому он применяется. Чтобы добавить модификатор точности, следует поставить за спецификатором ширины поля десятичную точку, а после нее – требуемое значение точности (число знаков после десятичной точки). Применительно к целым числам модификатор точности задает минимальное количество выводимых цифр. При необходимости перед целым числом будут добавлены нули. Если модификатор точности применяется к строкам, то число, следующее за точкой, задает максимальную длину поля. Например, спецификация %5.7s выведет строку длиной не менее пяти, но не более семи символов. Если выводимая строка окажется длиннее максимальной длины поля, конечные символы будут отсечены. По умолчанию все выводимые значения выравниваются по правому краю: если ширина поля больше выводимого значения, то оно будет выровнено по правому краю поля. Чтобы установить выравнивание по левому краю, нужно поставить знак "минус" ("–") сразу после знака процента. Например, спецификация формата %–10.4f обеспечит выравнивание вещественного числа с четырьмя десятичными знаками по левому краю в 10-символьном поле. Существуют два модификатора формата, позволяющие функции printf() отображать короткие

и длинные целые числа. Это модификатор l (латинская буква эль) уведомляет функцию printf() о длинном типе значения. Модификатор h сообщает функции printf(), что нужно вывести число короткого целого типа. Кроме того, модификатор l можно поставить перед командами форматирования вещественных чисел. В этом случае он уведомит о выводе знач ения типа long double.

Спецификаторы формата для функции printf() перечислены в табл. 2.5.

Таблица 2.5.

Спецификаторы формата функции printf()

Код	Формат
%c	Символ
%d	Десятичное целое число со знаком
%i	Десятичное целое число со знаком
%e	Экспоненциальное представление числа (в виде мантиссы и порядка, e — на нижнем регистре)
%E	Экспоненциальное представление числа (в виде мантиссы и порядка, E — на верхнем регистре)
%f	Десятичное число с плавающей точкой
%F	Десятичное число с плавающей точкой (только стандарт C99; если применяется к бесконечности или нечисловому значению, то выдает надписи INF, INFINITY(бесконечность) или NAN — Not A Number на верхнем регистре. Спецификатор %f выводит их эквиваленты на нижнем регистре)
%g	Использует более короткий из форматов %e или %f
%G	Использует более короткий из форматов %E или %F
%o	Восьмеричное число без знака
%s	Символьная строка
%x	Шестнадцатеричное без знака (строчные буквы)
%X	Шестнадцатеричное без знака (прописные буквы)
%p	Выводит указатель
%n	Соответствующий аргумент должен быть указателем на целое число. (Этот спецификатор указывает, что в целочисленной переменной, на которую указывает ассоциированный с данным спецификатором указатель, будет храниться число символов, выведенных к моменту обработки спецификации %n)
%%	Выводит знак процента

Прототип функции getchar() имеет следующий вид:

```
int getchar(void);
```

Функция getchar() возвращает из стандартного потока stdin (входного потока данных) следующий символ. При чтении символа предполагается, что символ имеет тип unsigned char, который потом преобразуется в целый. При достижении конца файла, как и при обнаружении ошибки, функция getchar()

возвращает значение EOF (End Of File – конец файла).

Прототип функции gets имеет следующий вид:

```
char *gets(char *str);
```

Функция gets() читает символы (включая пробелы) из стандартного потока stdin и помещает их в массив символов, адресуемый указателем *str (далее это массив символов). Символы читаются до тех пор, пока не встретится разделитель строк или значение EOF. Для реализации EOF на клавиатуре следует набрать одновременно Ctrl+Z. Вместо разделителя строк в конец строки вставляется нулевой символ, свидетельствующий о ее завершении. Следует учесть, что нет способа ограничить количество символов, которое прочитает функция gets(). Поэтому массив, адресуемый указателем *str, может переполниться, и тогда программа выдаст непредсказуемые результаты.

Прототип функции scanf() имеет следующий вид:

```
int scanf(const char *format, ?);
```

Функция scanf() представляет собой функцию для ввода данных общего назначения, которая читает поток stdin и сохраняет информацию в переменных, перечисленных в списке аргументов. Если в строке форматирования встретится разделитель, то функция scanf() пропустит один или несколько разделителей во входном потоке. Под разделителем, или пробельным символом, подразумевают пробел, символ табуляции \t или разделитель строк \n. Все переменные должны передаваться посредством своих адресов, например, с помощью символа &. Управляющая строка, задаваемая параметром format, состоит из символов трех категорий: спецификаторов формата, пробельных символов, символов, отличных от пробельных.

Спецификация формата начинается знаком % и сообщает функции scanf() тип данного, которое будет прочитано. Спецификации формата функции scanf() приведены в табл.2.6.

Таблица 2.6.

Спецификаторы формата функции scanf()

Код	Формат
%c	Читает один символ
%d	Читает десятичное целое число
%i	Читает целое число в любом формате (десятичное, восьмеричное или шестнадцатеричное)
%u	Читает десятичное целое число типа short int
%e	Читает число с плавающей точкой (и в экспоненциальной форме)
%E	Аналогично коду %e
%f	Читает число с плавающей точкой
%lf	Читает десятичное число с плавающей точкой типа double

Продолжение таблицы 2.6. – Спецификаторы формата функции scanf()

%F	Аналогично коду %f (для стандарта C99)
%g	Читает число с плавающей точкой.
%G	Аналогично коду %g
%o	Читает восьмеричное число
%x	Читает шестнадцатеричное число
%X	Аналогично коду %x
%s	Читает строку
%p	Читает указатель
%n	Принимает целое значение, равное количеству прочитанных до сих пор символов
%[ ]	Просматривает набор символов
%%	Читает знак процента

Строка форматирования читается слева направо, и спецификации формата сопоставляются с аргументом в порядке их перечисления в списке аргументов. Символ *, стоящий после знака % и перед кодом формата, прочитает данные заданного типа, но запретит их присваивание. Команды форматирования могут содержать модификатор максимальной длины поля. Он представляет собой целое число, располагаемое между знаком % и кодом формата, которое ограничивает количество читаемых для всех полей символов. Если входной поток содержит больше заданного количества символов, то при последующем обращении к операции ввода чтение начнется с того места, в котором "остановился" предыдущий вызов функции scanf() [2.3]. Если разделитель (например, пробел) встретится раньше, чем достигнута максимальная ширина поля, то ввод данных завершится. В этом случае функция scanf() переходит к чтению следующего поля. При чтении одиночных символов символы табуляции и разделители строк читаются подобно любому другому символу.

В программах бывает необходимость определять константы. В языке **C** типы констант можно задавать явно при использовании суффиксов. Например:

```
long int  j = -12345678L;       /* суффикс L */
unsigned int a = 678U;          /* суффикс U */
float x = 123.45F;              /* суффикс F */
long double z = 12345678.99L;   /* суффикс L* /
```

По умолчанию спецификации f, e, g заставляют функцию scanf() присваивать переменным типа float. Если перед одной из этих спецификаций поставить модификатор l, то функция scanf() присвоит прочитанные данные переменной типа double.

Функция scanf() поддерживает спецификатор формата общего назначения, называемый набором сканируемых символов. В этом случае определяется набор символов, которые могут быть прочитаны функцией scanf() и присвоены соответствующему массиву символов. Для определения

такого набора символы, подлежащие сканированию, необходимо заключить в квадратные скобки. Открывающая квадратная скобка должна следовать сразу за знаком процента. При использовании набора сканируемых символов функция scanf() продолжает читать символы и помещать их в соответствующий массив символов до тех пор, пока не встретится символ, отсутствующий в данном наборе. Если первый символ в наборе является знаком "^", то получится обратный эффект: входное поле читается до тех пор, пока не встретится символ из заданного набора сканируемых символов, т.е. знак "^" заставляет функцию scanf() читать только те символы, которые отсутствуют в наборе сканируемых символов. Если в строке форматирования встретился символ, отличный от разделителя, то функция scanf() прочитает и отбросит его. Если заданный символ не найден, то функция scanf() завершает работу.

В таких средах разработки как MS Visual Studio 2008 и 2010 рекомендуется для безопасной работы применять функции gets_s() и scanf_s(). Для этих функций при чтении символа или строки следует указать размер в байтах, соответственно для символа или строки. Например, scanf_s("%c", &ch, 1). В Visual Studio 2010 тип данных char занимает 1 байт.

Практическая часть.

Пример 1. *Напишите программу вычисления площади круга и его длины окружности по заданному радиусу, вводимого пользователем с клавиатуры, а также вывода на консоль максимальных значений чисел типа int, float и double.*

Для решения примера следует воспользоваться математической библиотекой компилятора, т. е. включить в программу заголовочный файл <math.h>., а также заголовочные файлы <limits.h>, <float.h>.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
// Для числа пи (π)
#define _USE_MATH_DEFINES
#include <math.h>
#include <limits.h>
#include <float.h>
int main (void)
{
double  R,  Sr, Lr;
printf("\n Enter a real greater than zero: ");
scanf_s("%lf", &R);
Sr = M_PI*R*R;
Lr = 2*M_PI*R;
printf("\n Area of a circle of radius R = %g is %g", R,
Sr);
```

```
printf("\n The length of a circle of radius R = %g is
%g",R,Lr);
puts("");
printf("\n  Maximum integer: %d\n ", INT_MAX);
printf("  Maximum real number of float:  %g\n  ",
FLT_MAX);
printf("Maximum real number type double:  %g\n  ",
DBL_MAX);

printf("\n Press any key: ");
      _getch();
      return 0;
}
```

В программу включена константа _USE_MATH_DEFINES для работы с числом M_PI (π). Остальные константы можно найти в справочной документации компилятора. Например, через меню Help →Index системы MS Visual Studio 2008.

Функция `scanf_s()` определена в компиляторе языка **C** системы MS Visual Studio 2008. С этой функцией компилятор не выдает предупреждений.

Результат выполнения программы показан на рис. 2.1 .

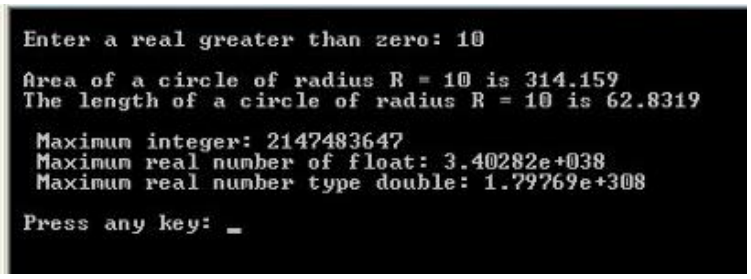


Рис. 2.1. Пример использования предопределенных констант

В начале функции `int main(void)` сделано объявление переменных, которые будут использоваться в программе. Каждый тип переменных объявлен через запятую.

Функции `printf()` выводят либо только сообщения, либо еще заданные переменные соответствующих типов.

Функция `gets()` позволяет считывать символы с наличием разделителей, в частности, с пробелами. В Microsoft Visual Studio 2010 рекомендуется использовать `gets_s()`, чтобы не было предупреждений.

Следует обратить внимание на формат записи функций scanf(). Если сканируются числа, или одиночные символы, то присваивание этих символов переменным, которые были объвлены через соответствующие типы данных, осуществляется с помощью взятия адреса этих переменных, т.е. с помощью символа &, например, scanf_s(«%c»,&ch, 1). При сканировании массива символов, т.е. при сканировании строки, символ & не используется. Имя массива само по себе является указателем. Обращение к адресу

осуществляется с помощью указателей (будут рассмотрены позднее). Для сканирования чисел типа double в функции scanf_s() используется спецификатор l.

2.2.1. Математические функции в языке программирования СИ++

Таблица 2.7.

Математическая функция	Название функции	Функция на языке программирования СИ++	Пояснение
arccos *x*	арккосинус	`#include <math.h> double acos(double x); float acosf(float x);`	аргумент для acos должен находиться в отрезке [-1,1]. Acos и acosf возвращают значения в радианах на промежутке от 0 до pi.
Arccosh(x)	обратный гиперболический косинус	`#include <math.h> double acosh(double x); float acoshf(float x);`	x должен быть больше либо равен 1.
arcsin *x*	арксинус	`#include <math.h> double asin(double x); float asinf(float x);`	аргумент для asin должен находиться в отрезке [-1,1]. Asin и asinf возвращают значения в радианах в промежутке от –pi/2 до pi/2.
Arcsinh(x)	обратный гиперболический синус	`#include <math.h> double asinh(double x); float asinhf(float x);`	ни atanh, ни atanhf не являются ANSI C – функциями.
arctgx	арктангенс	`#include <math.h> double atan(double x); float atanf(float x);`	atan и atanf возвращают значения в радианах на промежутке от –pi/2 до pi/2.

Продолжение таблицы 2.7. – Математические функции в языке программирования СИ++

$arctgh(x)$	обратный гиперболический тангенс	`#include <math.h> double atanh(double x); float atanhf(float x);`	ни atanh, ни atanhf не являются ANSI C - функциями.
$\sqrt[3]{x}$	кубический корень	`#include <math.h> double cbrt(double x); float cbrtf(float x);`	Является стандартной функцией ANSI C
$\cosh x$	гиперболический косинус	`#include <math.h> double cosh(double x); float coshf(float x);`	Углы определены в радианах.
e^x	экспонента числа	`#include <math.h> double exp(double x); float expf(float x);`	Является стандартной функцией ANSI C
$\|x\|$	модуль числа (абсолютная величина)	`#include <math.h> double fabs(double x); float fabsf(float x);`	Является стандартной функцией ANSI C
Min и max	наименьшее и наибольшее ближайшие целые	`#include <math.h> double floor(double x); float floorf(float x); double ceil(double x); float ceilf(float x);`	Является стандартной функцией ANSI C

Продолжение таблицы 2.7. – Математические функции в языке программирования СИ++

mod	остаток от деления в виде числа с плавающей точкой	`#include <math.h> double fmod(double x, double y); float fmodf(float x, float y);`	Является стандартной функцией ANSI C
$\ln x$	натуральный логарифм	`#include <math.h> double log(double x); float logf(float x);`	Является стандартной функцией ANSI C
$\lg x$	логарифм по основанию 10	`#include <math.h> double log10(double x); float log10f(float x);`	log10 возвращает значение логарифма по основанию 10 от x. Он определяется как ln(x)/ln(10).
x^y	возведение основания ***x*** в степень ***y***	`#include <math.h> double pow(double x, double y); float powf(float x, float y);`	Является стандартной функцией ANSI C
rint, rintf,	округление до ближайшего целого	`#include <math.h> double rint(double x); float rintf(float x);`	Является стандартной функцией ANSI C
	остаток от деления x/y	`double remainder(double x, double y); float remainderf(float x, float y);`	это будет число между – y/2 и y/2.

Продолжение таблицы 2.7. – Математические функции в языке программирования СИ++

$\sqrt{x}$	квадратный корень из числа	`#include <math.h> double sqrt(double x); float sqrtf(float x);`	вычисляет арифметический (неотрицательный) квадратный корень из аргумента
$\sin x$	синус	`#include <math.h> double sin(double x); float sinf(float x);`	Углы определены в радианах.
$\cos x$	косинус	`#include <math.h> double cos(double x); float cosf(float x);`	Углы определены в радианах.
$\sinh x$	гиперболический синус	`#include <math.h> double sinh(double x); float sinhf(float x);`	Углы определены в радианах.
$tg(x)$	тангенс	`#include <math.h> double tan(double x); float tanf(float x);`	Углы определены в радианах.
$tgh(x)$	гиперболический тангенс	`#include <math.h> double tanh(double x); float tanhf(float x);`	Углы определены в радианах. tanh(x) определяется как sinh(x)/cos(x)

Пример 2. *Создать блок-схему к программе и программу на языке программирования Си++ для вычисления функции* ***B****, которая зависит от трех переменных* ***x, y, z****. Ввод значений переменных сделать с клавиатуры, если* $B = x^2 + 3xy + \sqrt{z}$, где $z = (x + y)^3$

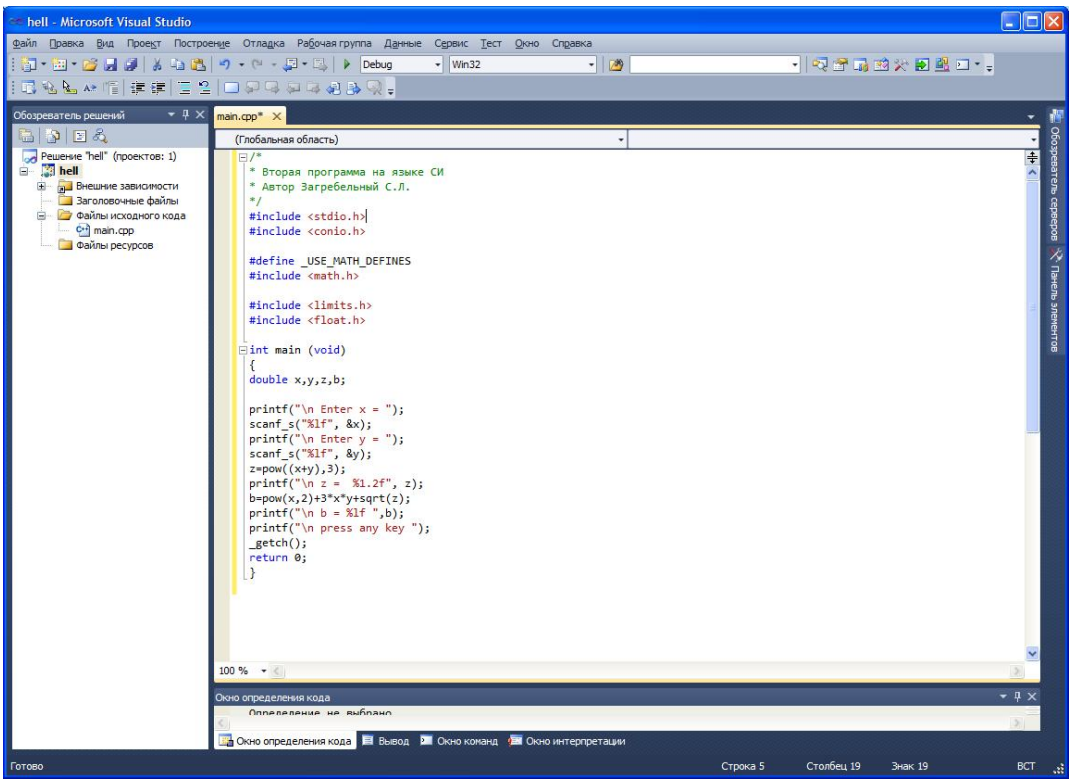


Рис. 2.2. Окно программы Примера 2

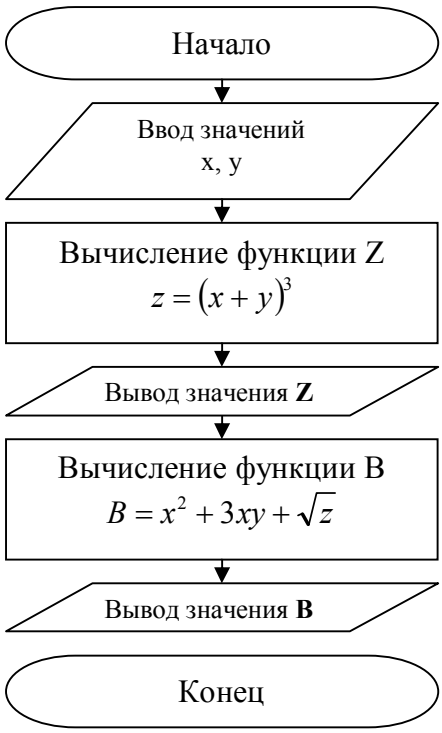
Программный код решения примера:

```
/*
* Вторая программа на языке СИ
* Автор Загребельный С.Л.
*/
#include <stdio.h>
#include <conio.h>

#define _USE_MATH_DEFINES
#include <math.h>

#include <limits.h>
#include <float.h>

int main (void)
{
double x,y,z,b;
printf("\n Enter x = ");
scanf_s("%lf", &x);
printf("\n Enter y = ");
scanf_s("%lf", &y);
z=pow((x+y),3);
printf("\n z =  %1.2f", z);
b=pow(x,2)+3*x*y+sqrt(z);
printf("\n b = %lf ",b);
```



```
printf("\n press any key ");
_getch();
return 0;
}
```

В программу включена библиотека математических функций math.h, в которой sqrt() – функция извлечения квадратного корня, pow(*a, b*) –функция возводит в степень *b* число по основанию *a*. Все эти функции возвращают результат типа double и вычисляют от числа (выражения) также типа double.

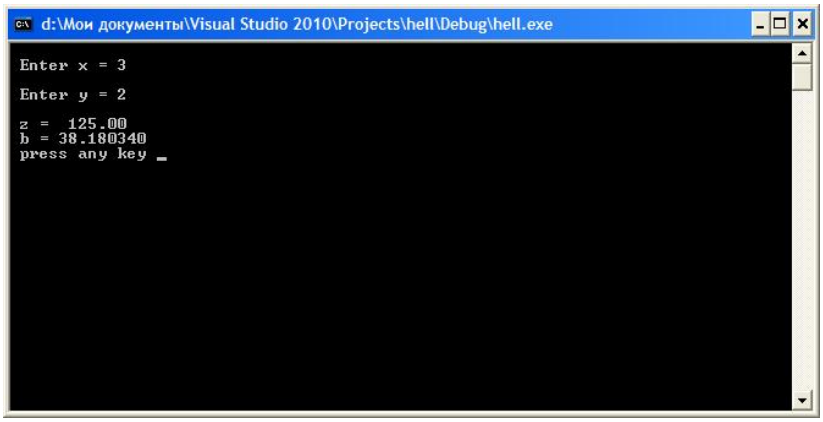


Рис. 2.3. Окно вывода результата Примера 2

Индивидуальные задания

Составить блок-схему к программе и программу на языке программирования СИ++ для вычисления функций b=f(x,y,z) , где z=w(x,y) при постоянных значениях x и y (см. пример 2). Значения x и y заданы в таблице 2.8.

Таблица 2.8.

Вариант	*f(x,y,z)*	*w(x,y)*	*x*	*y*
1	$e^{-2x}(tg(z)+2y)$	$\sqrt{\sin^2\lvert x\rvert+y}$	-4,52	0,75
2	$\frac{\sqrt{x}\sin(2y)}{z+e^{z}y}$	$\frac{2xy}{x+\cos(y)}$	2,87	0,84
3	$\frac{y-z/(y-x)}{\cos(x)+(y-x)^2}$	$\frac{\sqrt{15y}}{y+ctg(x)}$	1,82	18,25
4	$y^{X}+\sqrt[3]{\lvert x\rvert+\lvert y\rvert}e^{Z}$	$\frac{\sqrt{\lvert 20x\rvert}}{x^2+y^3}$	-0,85	1,25
5	$\ln(\sqrt{x}-\sqrt{y}+2)z^3$	$\frac{\sin(x/y)}{2x^2}$	25,34	33,85
6	$y+\frac{xarctg(z)}{y+x^2}$	$\sqrt{x}\sin(y)$	0,12	-8,75
7	$\frac{z^3}{x+y^3/(x+z^2)}$	$\frac{15}{x+e^{y}}$	1,54	3,26

Продолжение таблицы 2.8.

8	$\frac{z^2}{y+x^3}+\sin(y/5)$	$\frac{3x}{\cos^2(y)+3}$	1,58	3,42
9	$\lvert\cos(x)+\sin(y)\rvert-2tg^2(z)$	$\frac{\sqrt{x}\sin^2(y)}{x+e^y}$	0,42	-0,87
10	$\ln y^{\sqrt{\lvert x\rvert}}(z^2-\frac{y}{\sin(x)})$	$\sqrt{3+2y}$	-15,24	4,67
11	$\frac{z^2}{y+x^3}+\sin(y/5)$	$\frac{\sqrt{x}arctg(2y)}{e^{y+x}}$	6,55	-2,78
12	$\cos^2(z)+\lvert x+y\rvert^3$	$\frac{12}{x+e^y}$	-2,75	-1,42
13	$x^{y/x}+\sqrt[3]{\lvert y^z\rvert}e^x$	$\ln(\sqrt{e^{x-y}}+z^2)$	1,82	18,23
14	$\frac{e^{z-1}}{2y+x^3}+\sin(y^2)$	$\cos^2(y)+\sin^3(x^2)$	0,84	0,65
15	$\sqrt{\lvert y\rvert e^{-(y+x)}}-\cos(z^3)$	$\frac{x+6y}{\sin(x)+\ln(y)}$	1,12	0,87
16	$\frac{4y^2e^{3x}}{8z^3+\ln\lvert x\rvert}$	$\frac{x+y\sqrt{x}}{x+10}$	0,27	4,38
17	$\frac{\sqrt{y\ln(x)}-zx^2}{1+tg^2(x^2)}$	$\frac{e^x\sqrt{x^3+y}}{x-1}$	6,35	7,32
18	$\frac{\ln(y+\sqrt{y+x^2})}{(z+x^2)e^{x/2}}$	$\frac{2x\sqrt{y}}{\sin(x^2)}$	0,42	1,23
19	$\frac{x^3+y}{\sin^2(z)+x/5}$	$\frac{\cos^2(3(2+x))}{4-y^2\sqrt{x}}$	43,32	-0,54
20	$\frac{x+y(x^2+\cos(y))}{y(x-z)+\ln\lvert xz\rvert}$	$2\sin(3x+y)$	3,25	4,12
21	$\frac{1+\cos^2(x+z)}{\lvert x^3-2\ln\sqrt{y}\rvert}$	$\frac{x^2+y^2}{e^{x+y}}$	0,83	2,38
22	$\frac{\ln\lvert x\rvert}{\sqrt[3]{\lvert x\rvert+\lvert y\rvert}+tg(z)}$	$\sqrt{x^2-\sin(y)}$	-0,93	-0,25
23	$\frac{z^3}{x+y^3/(x+z^2)}$	$\frac{\lvert y+8x\rvert}{\sin(x)+tg(y)}$	-0,72	-1,42
24	$2^{-x}\sqrt{y+\sqrt[4]{\lvert z\rvert}}$	$\frac{xy}{x^2+5}+\cos^2(y)$	3,98	1,63
25	$\sqrt{e^{x-1}\sqrt{\lvert y\rvert}}$	$\frac{3y}{3+e^{x-y}}$	3,91	-0,51

26	$\frac{\sqrt[3]{\sqrt{xy}-x^2}}{1+z^2}$	$\frac{x^3+\sqrt{xy}}{(x+y^2)}$	1,26	3,69
27	$\frac{\ln\lvert z^3+x+y\rvert}{\sqrt{x^2+y^2}-\sqrt{z}}$	$\frac{y\sqrt{(x+y)^4}}{x+10y}$	-4,11	2,99
28	$\frac{(x+y+z^3)^2}{(x+z)/z^2}$	$\frac{x^2\sqrt{y}-y^2\sqrt{x}}{x^2+y^2}$	1,24	2,55
29	$\frac{\sqrt[3]{tg(x+y+z)}}{\ln\lvert x+y-z\rvert}$	$\frac{x-\sqrt[3]{y}}{y+\sqrt[3]{x}}$	-1,25	-3,16
30	$\frac{\sqrt[3]{\cos(x)+\sin(y)}}{tg(z)}$	$\frac{x^2+x^y}{\cos(\sqrt[3]{x})}$	1,84	-1,17

Контрольные вопросы

1. Для каких типов данных используются суффиксы при инициализации переменных?
2. Чем отличаются функции `printf()` и `puts()` при консольном выводе информации?
3. Для чего в программах на **C++** используется заголовочный файл math.h?
4. При использовании функции `gets_s()` с какими разделителями может происходить считывание информации с консоли?
5. Какой тип данных возвращает функция `gets_s()` при считывании информации?
6. Как осуществляется считывание с консоли информация с помощью функции `scanf_s()`?
7. Как с консоли осуществляется считывание последовательности различных типов данных с помощью одной функции `scanf_s()`?
8. Как выводится на консоль последовательность различных типов данных с помощью одной функции `printf()`?

ЛАБОРАТОРНАЯ РАБОТА 3

Принятие решений. Условные операторы в языке C++.

Теоретическая часть

В языке программирования **C** используются несколько конструкций для принятия решений:

- оператор if;
- оператор switch;
- условный оператор ? (оператор условия).

Для прерывания программного цикла при некотором условии применяется утверждение (оператор) break, для продолжения итераций цикла при выполнении некоторых условий применяется утверждение (оператор) continue, для выхода из функции при выполнении некоторых условий применяется оператор return, для перехода к заданному месту программы применяется оператор goto, хотя считается, что в программировании не существует ситуаций, в которых нельзя обойтись без оператора goto [2; 3]. Утверждение break применяется также в теле оператора switch.

3.1. Оператор if

Общая форма записи оператора if:

```
if (expression)
program statement;
```

В операторе if используется результат вычисления условия, заключенного в круглые скобки, на основе которого принимается решение. Результат вычисления условия expression может быть арифметическим или логическим. Если результат выполнения условия expression будет истинным, то возможно выполнить несколько утверждений типа program statement. Для этого следует использовать фигурные скобки, например:

```
if (expression)
{
program1 statement1;
program2 statement2;
...
}
```

3.2. Конструкция if–else

Общая форма записи конструкции if–else:

```
if (expression)
program1 statement1;
else
program2 statement2;
```

Если выполняется условие expression, то будет выполняться фрагмент программы program1 statement1, в противном случае будет выполняться program2 statement2.

Каждое из утверждений может быть множественным. В таком случае применяются фигурные скобки:

```
if (expression)
{
program1 statement1;
program2 statement2;
...
}
else
{
program33 statement33;
program34 statement34;
...
}
```

3.3. Конструкция if–else if–else if–...–else

Форма записи конструкции if–else if–else if–...–else:

```
if (expression1)
program1 statement1;

else if (expression2)
program2 statement2;

else if (expression3)
program3 statement3;
...
else
program statement;
```

Приведенная конструкция используется для выбора возможных ситуаций, когда проверяются условия expression1, expression2, expression1,... . Соответственно будут выполняться действия program1 statement1, program2 statement2, program3 statement3 и т.д. В случае, когда ни одно из условий не выполняется, выполняются действия, прописанные после оператора else.

В случае выполнения множественных действий применяются фигурные скобки для каждого из утверждений:

```
if (expression1)
{
program1 statement1;
...
}
```

```
else if (expression2)
{
program2 statement2;
...
}

else if (expression3)
{
program3 statement3;
...
}
...

else
{
program statement;
...
}
```

3.4. Оператор switch

Общая форма записи оператора switch:

```
switch (expression) {
case value1:
program statement;
...
break;

case value2:
program statement;
...
break;
...

case valuen:
program statement;
...
break;

default:
program statement;
...
break;
}
```

Выражение заключенного в круглые скобки оператора

последовательно сравнивается со значениями value1, value2,..., valuen, которые должны быть простыми константами или константными выражениями. В том случае, когда одно из этих значений равно значению, выполняются утверждения, которые следуют за данным значением.

Утверждение break сигнализирует об окончании выполнения утверждений и приводит к выходу из оператора switch. Утверждение break ставится в конце каждого варианта выбора. Если этого не сделать, то выполнение последовательности утверждений перейдет в следующий вариант выбора и будет выполняться до тех пор, пока не встретится утверждение break.

Специальный дополнительный вариант default будет выполнен в том случае, когда не будет найдено ни одного совпадения.

Операторы `if` и `switch` той или иной синтаксической конструкции существуют практически во всех языках программирования (в первую очередь языках высокого уровня), и их часто называют операторами ветвления.

3.5. Условный оператор

В отличие от других операторов языка **C**, которые могут быть унарными или бинарными, специфический оператор условия является тернарным оператором. Это означает, что у него может быть три операнда.

Общий формат записи оператора условия:

```
условие ? выражение_1 : выражение_2
```

Если в результате вычисления **условия** будет получено значение TRUE (истина, не нуль), то выполняется выражение_1, и результатом выполнения оператора условия будет значение, полученное при вычислении этого выражения. Если в результате вычисления условия будет получено значение FALSE (ложь, т.е. нуль), то выполняется выражение_2, и результатом выполнения оператора условия будет значение, полученное при вычислении выражение_2.

Оператор условия часто описывают как оператор ?. Тернарный оператор условия ? наиболее часто используется для присвоения переменной одного из двух значений в зависимости от некоторого условия.

3.6. Оператор break (от английского – прерывать)

Оператор или утверждение break служит для немедленного выхода из цикла, будь то while, for или do–while. После выхода из цикла выполнение программы продолжается с утверждения (фрагмента программы), непосредственно следующего за циклом.

Если оператор break встречается во вложенном цикле (вложенных циклах), то будет прекращено выполнение того цикла, в котором этот оператор встретился.

Необходимость в использовании оператора прерывания break в теле цикла возникает тогда, когда условие продолжения итераций нужно проверять не в начале цикла (как в циклах while и for) и не в конце тела цикла

(как в цикле do–while), а в середине тела цикла.

Формат записи оператора break:

```
break;
```

3.7. Оператор continue (от английского – продолжать)

Оператор или утверждение continue служит для перехода к следующей итерации цикла.

Оператор continue противоположен по действию оператору break. Оператор continue позволяет в любой точке тела цикла (while, for или do–while) прервать текущую итерацию и перейти к проверке условий продолжения цикла. В соответствии с результатами проверки либо заканчивается выполнение цикла, либо начинается новая итерация. При этом все утверждения (фрагменты программы), которые следуют за оператором continue (ключевым словом), автоматически пропускаются.

Формат записи оператора continue:

```
continue;
```

3.8. Оператор goto

Сейчас во многих языках программирования оператор безусловного перехода типа goto не используется. Однако в языке программирования **C** он имеет место. Применение оператора goto не является хорошим стилем программирования. Но в некоторых случаях его применение бывает уместно. Иногда, при умелом использовании, оператор goto может оказаться весьма полезным, например, если нужно покинуть глубоко вложенные циклы.

Для оператора goto всегда необходима метка. Метка – это идентификатор с последующим двоеточием. Метка должна находиться в той же функции, что и оператор goto, переход в другую функцию невозможен.

Общий формат записи оператора goto:

```
goto метка;
.
.
.
метка: заданные действия.
```

Метка может находиться как до, так и после оператора goto. С помощью оператора goto можно не только выходить из цикла, но и организовать цикл.

Логические операторы отношения приведены в табл. 3.1.

Таблица 3.1.

№ п/п	Оператор	Операция
1)	&&	И
2)	||	ИЛИ
3)	!	НЕ, отрицание

Ниже приведены операции отношений в убывающей

последовательности приоритетов:

```
Наивысший          !
                  >   >=   <   <=
                  ==   !=
                  &&
Низший           ||
```

Как и в арифметических выражениях, для изменения порядка выполнения операций сравнения и логических операций можно использовать круглые скобки.

Операторы отношения перечислены в табл. 3.2.

Таблица 3.2.

Операторы отношения языка программирования С

№ п/п	Оператор	Значение
1)	==	Равно
2)	!=	Не равно
3)	<	Меньше
4)	<=	Меньше или равно
5)	>	Больше
6)	>=	больше или равно

Результат любой операции сравнения или логической операции есть **0** (нуль) или **1**.

Практическая часть.

Пример 1. *Напишите программу решения квадратного уравнения с проверкой на наличие вещественных (не комплексных) корней на основе только операторов if. Вид квадратного уравнения:*

$ax^2 + bx + c = 0$

Как известно, квадратное уравнение будет иметь вещественные корни, если его дискриминант будет неотрицательным, т.е. когда

$D = b^2 - 4ac \geq 0$

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int main(void) {
      float a, b, c;
      float D, x1, x2, x;
      printf("\n\t Equation a*x^2 + b*x + c = 0\n");
      printf("\n\t Enter the coefficient a: ");
      scanf_s("%f", &a);
      printf("\t Enter the coefficient b: ");
```

```
      scanf_s("%f", &b);
      printf("\t Enter the coefficient c: ");
      scanf_s("%f", &c);
      D = b*b - 4*a*c;
      if (D >= 0 && a != 0) {
      x1 = -b/(2*a) + (float)sqrt(D)/(2*a);
      x2 = -b/(2*a) - (float)sqrt(D)/(2*a);
     printf("\n\t The roots of the equation:\n\t x1 =
%1.4f, x2 = %1.4f\n", x1, x2);
      }
      if (D < 0)
            printf("\n\t The roots of complex\n");

      if (a == 0 && b != 0) {
x = -c/b;
      printf("\n\t As a = %1.0f,\n\t the solution of the
equation is: %1.4f\n", a, x); }

      printf("\n Press any key: ");
      _getch();
      return 0;
}
```

Возможный результат выполнения программы показан на рис. 3.1.

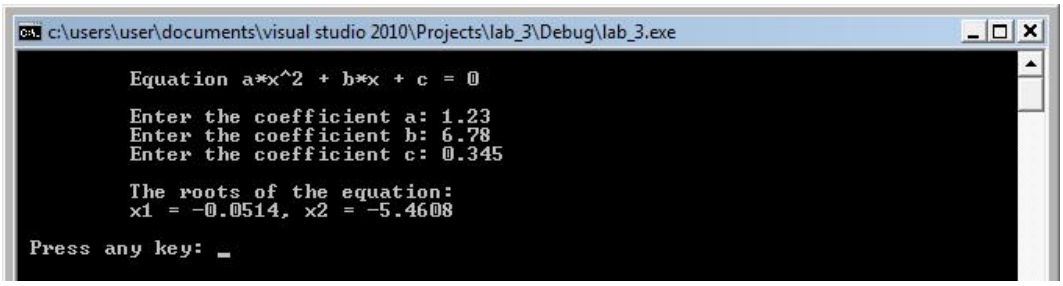


Рис. 3.1. Результат решения квадратного уравнения

В программе последовательно проверяются условия с помощью операторов if. В последнем случае, когда коэффициент a = 0, квадратное уравнение вырождается и превращается в линейное уравнение. Решение в этом случае очевидно.

В программе применены функции scanf_s() вместо стандартной функции scanf() языка **C**. Это сделано для того, чтобы по этим функциям не было предупреждений (warning) в MS Visual Studio 2008.

Кроме того, в программу подключена библиотека math.h для действий с математическими функциями, например, sqrt().

В первом операторе if применено логическое условие **И** (&&) для проверки того, что дискриминант не равен отрицательному значению и одновременно чтобы первый коэффициент квадратного уравнения не был равен нулю. Аналогичное условие прописано и для последнего оператора if.

Пример 2. *Напишите программу решения квадратного уравнения с проверкой на наличие вещественных корней на основе конструкции if–else. Вид квадратного уравнения:*

$$ax^2 + bx + c = 0$$

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int main(void)
 {
      float a, b, c;
      float D, x1, x2, x;
      printf("\n\t Equation a*x^2 + b*x + c = 0\n");
      printf("\n\t Enter the coefficient a: ");
      scanf_s("%f", &a);
      printf("\t Enter the coefficient b: ");
      scanf_s("%f", &b);
      printf("\t Enter the coefficient c: ");
      scanf_s("%f", &c);
      D = b*b - 4*a*c;
      if (D >= 0 && a != 0 && b != 0) {
      x1 = -b/(2*a) + (float)sqrt(D)/(2*a);
      x2 = -b/(2*a) - (float)sqrt(D)/(2*a);
     printf("\n\t The roots of the equation:\n\t x1 =
%1.4f, x2 = %1.4f\n", x1, x2);
      }
      else {
if (a == 0 && b != 0) {
x = -c/b;
if (c != 0)
printf("\n\t As a = %1.0f,\n\t the solution of the
equation is: %1.4f\n", a, x);


else
printf("\n\t As a = %1.0f and c = %1.0f,\n\t the
solution of the equation is: %1.0f\n", a, -x);
}
if (D < 0)
printf("\n\t The roots of complex\n");
      }

      printf("\n Press any key: ");
      _getch();
```

```
      return 0;
}
```

В программе использованы вложенные операторы if.

Результат выполнения программы при исключительной ситуации показан на рис. 3.2.

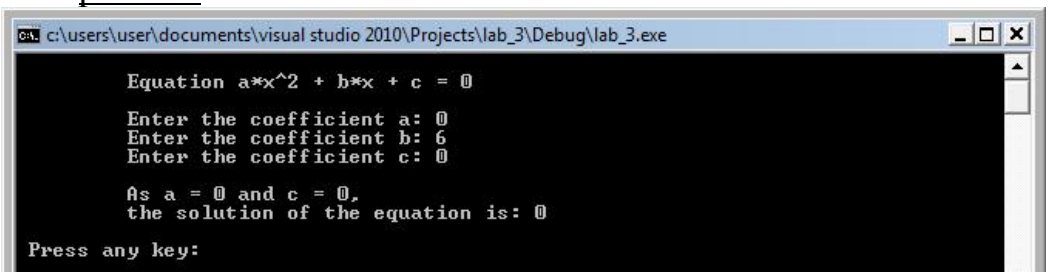


Рис. 3.2. Выполнение программы с двумя нулевыми коэффициентами

Пример 4. *Напишите программу расчета простого арифметического выражения на основе оператора switch.*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

int main (void) {
      float value1, value2;
      char operat;

     printf("\n\t Printed on the keyboard expression:
");
scanf_s("%f%c%f", &value1, &operat, sizeof(char),
&value2);

     switch (operat) {

     case '+':
           printf("\n\t Result: %1.4f\n", value1 +
value2);
           break;

     case '-':
           printf("\n\t Result: %1.4f\n", value1 -
value2);
           break;
      case '*':
           printf("\n\t Result: %1.4f\n", value1 *
value2);
           break;

     case '/':
```

```
            if (value2 == 0.0)
            printf("\n\t Division by zero.\n");
            else
           printf("\n\t Result: %1.4f\n", value1 /
value2);
            break;

     default:
            printf("\n\t Unknown arithmetic operator\n\t
error or enter a number. Break!\n");
            break;

      } // End switch

      printf("\n Press any key: ");
      _getch();
      return 0;
}
```

В программе использована полная форма оператора switch. Оператор break инициирует немедленный выход из оператора switch. Возможно использование вложенных операторов switch.

Возможный результат выполнения программы показан на рис. 3.3.

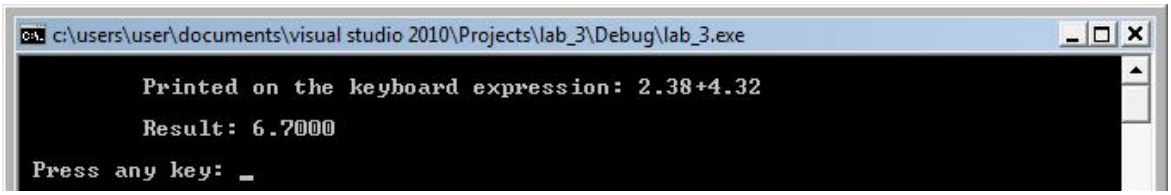


Рис. 3.3. Расчет простого арифметического выражения

Пример 5. *Напишите программу вычисления двух целых случайных чисел и определения наибольшего из них. Определение наибольшего числа произведите с помощью оператора условия ?*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h> // Для функций случайных чисел
#include <time.h>
int main (void) {
int a, b, maxab;
unsigned int some;
long int L;
L = (long) time(NULL);
some = (unsigned) L/2;
srand(some);
a = rand();
```

```
b = rand();
printf("\n\t Random numbers: a = %d; b = %d\n", a, b);
// Оператор условия для определения максимального числа
maxab = (a > b) ? a : b;
printf("\n\t Maximum number: %d\n", maxab);
      printf("\n Press any key: ");
      _getch();
      return 0;
 }
```

В программе использованы функции генерации псевдослучайных чисел rand() и задания исходного псевдослучайного числа srand(). Указанные функции входят в стандартную библиотечную функцию stdlib.h. Функция time() входит в библиотечную функцию time.h, которая поддерживает функции, обращающиеся к системному времени.

Для переменных L и some выполнено приведение типов.

При каждом обращении к функции rand() возвращается целое в интервале между нулем и значением RAND_MAX, которое в любой реализации должно быть не меньше числа 32767

Возможный результат выполнения программы показан на рис. 3.4.

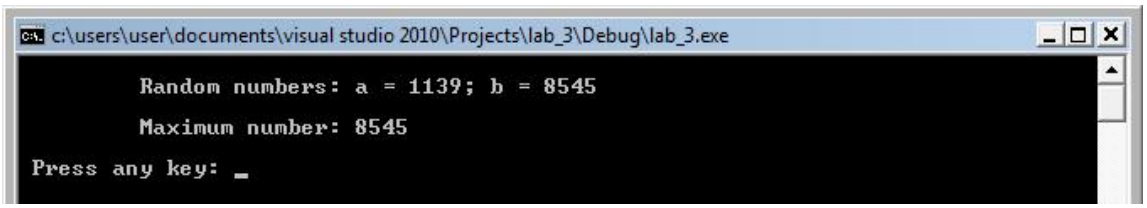


Рис. 3.4. Результат определения максимального числа

Пример 7. *Напишите программу распечатки четных целых чисел от 0 до 30.*

```
#include <stdio.h>
#include <conio.h>
int main (void)
 {
      int x;
      printf("\n\t Even numbers from 0 to 30:\n\n");
          for (x = 0; x < 31; x++)  {
      if ( x % 2 ) continue;
     printf("\t\t %3d\n", x);
      }
      printf("\n Press any key: ");
      _getch();
      return 0;
}
```

В программе в качестве проверки условия использовано деление по модулю (**x%2**). Если остаток от деления числа **x** не равен нулю, то

утверждение (оператор, инструкция) continue передает управление непосредственно инструкции, проверяющей условное выражение, после чего циклический процесс продолжается. С помощью программы выводятся только четные числа, а при обнаружении нечетного числа происходит преждевременный переход к следующей итерации цикла, и функция printf() опускается. Функция printf() включена в тело цикла оператора for.

Результат выполнения программы показан на рис. 3.5.

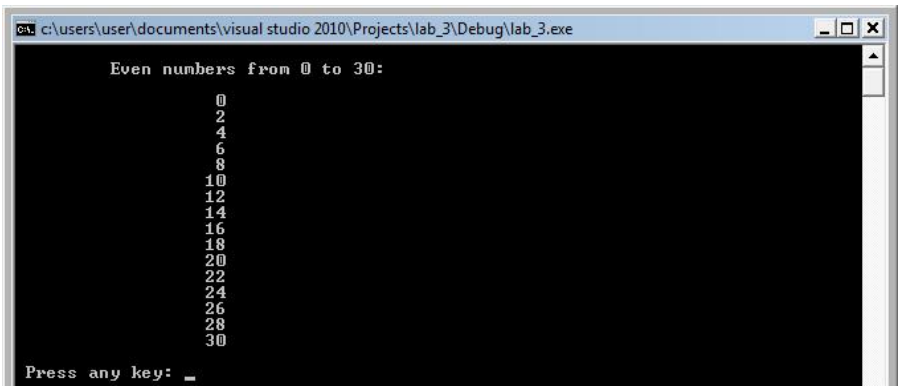


Рис. 3.5. Результат вывода четных чисел

Примечание. Оператор `goto` нельзя применять для перехода в тело цикла, т.е. метка не должна быть внутри оператора цикла. Метка может появиться текстуально до или после оператора goto.

Пример 8. *Создать блок-схему и программу вычисления функции с использованием оператора IF. Значение* ***x*** *и* ***a*** *ввести с клавиатуры.*

$$y = \begin{cases} \sqrt{2x^3 + 4x + 7}, & x \geq 0 \\ \dfrac{3x - 1}{(x^2 + 2)^2}, & x < a \end{cases}$$

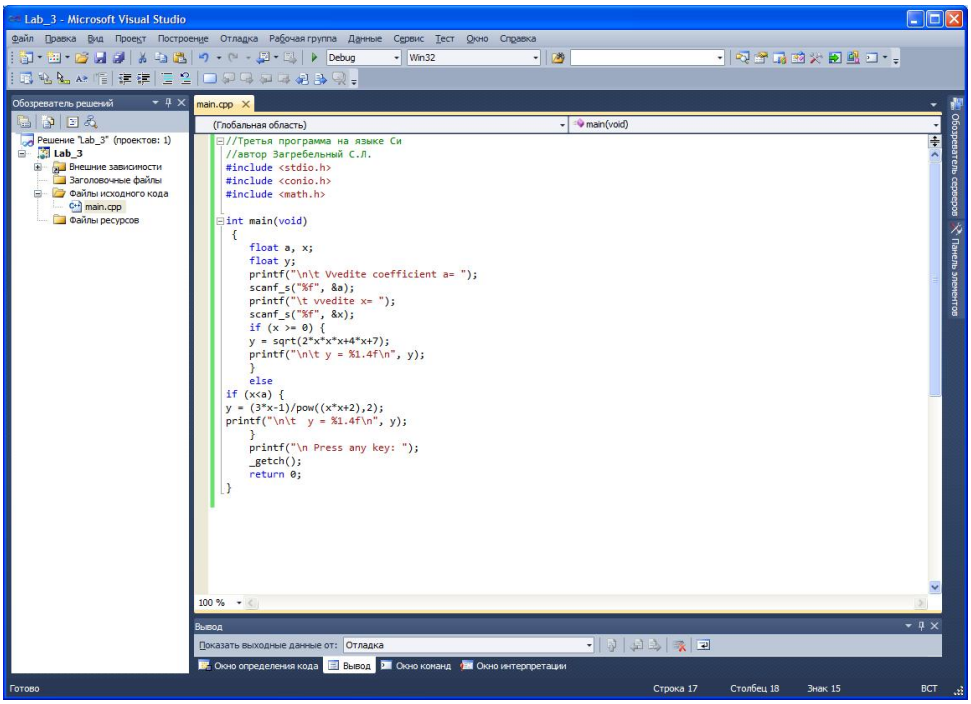
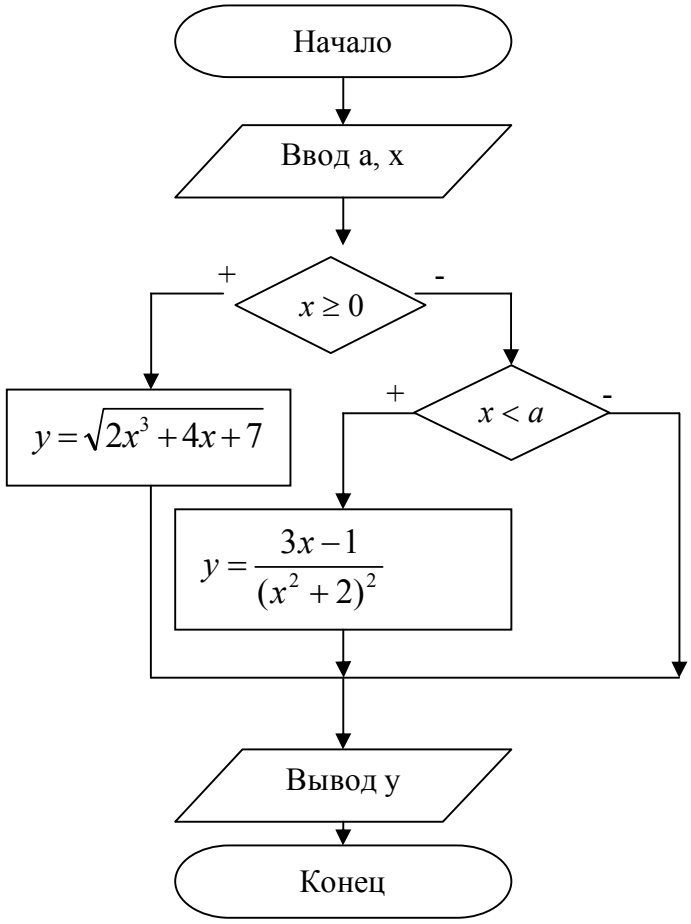


Рис. 3.6. Окно с программой




```
//Третья программа на языке Си
//автор Загребельный С.Л.
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main(void)
 {  float a, x;
    float y;
    printf("\n\t Vvedite coefficient a= ");
    scanf_s("%f", &a);
    printf("\t vvedite x= ");
    scanf_s("%f", &x);
    if (x >= 0) {
    y = sqrt(2*x*x*x+4*x+7);
    printf("\n\t y = %1.4f\n", y);
    }
    else
if (x<a) {y = (3*x-1)/pow((x*x+2),2);
printf("\n\t  y = %1.4f\n", y);}
    printf("\n Press any key: ");
    _getch();
    return 0;
}
```

Индивидуальные задания

Создать блок-схему и программу вычисления функции с использованием оператора IF. Вычислить значение функции ***y*** *для разных значений аргумента* ***x****,*

$$y=\begin{cases} f_1(x), если & x \le 0 \\ f_2(x), если & 0 < x \le a \\ f_3(x), если & x > a \end{cases}$$

значение ***x*** *и* ***a*** *вводить с клавиатуры.*

Вид функций $f_1(x)$, $f_2(x)$ и $f_3(x)$ выбрать из таблицы 3.2. в соответствии с номером своего варианта.

Таблица 3.2.

Вариант	$f_1(x)$	$f_2(x)$	$f_3(x)$
1	$\left\|\sqrt[3]{\dfrac{2x+5}{(x^3+2)}}\right\|$	$\dfrac{5x+x^2}{(x^2+3)^3}$	$\cos^2(x^3+\sqrt{x})$
2	$\sqrt[5]{x^2+x+1}$	$\ln^2(\lvert\sqrt{x+5}\rvert)$	$\sin(x^2)+x^{0,25}$
3	$\sqrt[3]{\lvert x\rvert+2}-1$	$\sin(x^3)+x^{0,5}$	$\ln^2(x)+\sqrt{x}$
4	$\sqrt{\sin^2 x+\cos^4 x}$	$\ln^2(x)+\sqrt{x}$	$\operatorname{tg}^2(x)+\sqrt{x}$
5	$x^3-\ln(\lvert x\rvert+1)$	$\dfrac{2x+2}{(\operatorname{tg}(2x-1)+1)}$	x^4-x^x
6	x^4+2x^3-x	$e^{-x}+\sqrt[4]{x}$	$\ln(x^3+x^2)$
7	$\dfrac{(3x-1)^2}{x^5}$	$\ln^2\lvert\sqrt{x+5}\rvert$	$\cos\left(\sqrt{1+x^2}\right)$
8	$x^5\operatorname{ctg}(2x^3)$	$\dfrac{5}{\operatorname{tg}(2x+3)+1}$	$\operatorname{tg}(x^2+1)e^{-x}$
9	$\lvert x\rvert\sin(3x)$	$x^3\cos(x+2)$	$\sin x^2+x^{0.25}$
10	$\lvert x\rvert^{2x+1}$	$\sin x^2$	$\ln^2\lvert x\rvert+\sqrt{x}$
11	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$2\sin(x-e^{-x})$
12	$2xe^{-x}$	$(x-1)^3+\cos(x^3)$	$2\sqrt{x^3}\sin(x^3)$
13	$\ln(x^2+5)$	$\sin(e^x+2)$	$\operatorname{tg}(5x+1)$
14	$2\sqrt{\lvert x^3\rvert}\sin(x^3)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$
15	$\cos x+x^3$	$\sqrt{x^3}\sin x$	$8+\cos(3x)$
16	$x\sin(x+4)$	$\ln(4x^2+1)$	$\ln\sqrt[5]{5+x^2}$

Продолжение таблицы 3.2.

17	x^4+2x^3-x	$1{,}3\sqrt{4+x^2}$	$\lvert x+1\rvert^x$
18	$\lvert x\rvert^5 \operatorname{ctg}\lvert x\rvert$	$\ln(x^2+1)$	$e^{-2x}-\sqrt[3]{\lvert x+1\rvert}$
19	$x^5\operatorname{ctg}(2x^3)$	$\sqrt[5]{x^4+3}$	$\lvert\sin^2 x+1\rvert^{2x}$
20	$\operatorname{ctg}(3x-1)^2$	$2+xe^{-x}$	$\sin^3 x^2$
21	$x\sin(x-1)$	$(x-1)^3+\cos x^3$	$\sqrt{\lvert x\rvert^3}\sin x^3$
22	$(x+1)/(\lvert x\rvert+2)^3$	$e^x+\cos(x+2)$	$3\ln\sqrt[5]{\sin^2 x+2}$
23	$3x^5-\operatorname{ctg} x^3$	$\ln(\sin 4x+1)^2$	$\sqrt[3]{2x^2+x^4+1}$
24	$1{,}3\sqrt{4+x^2}$	3^{x+3}	$x^{x+1}\sin(x+2)$
25	$e^{-3x}+\cos x$	$\sin^3 x^4$	$e^{-x}+\sqrt[3]{3x^2+1}$
26	$x^3+(\lvert x\rvert+1)^{0,1x}$	$(x-1)^3+\cos(2x^3)$	$\sin(7x)+\operatorname{tg}(0{,}01x)$
27	$\operatorname{tg}(0{,}1\pi x^2)+x$	$e^{x+1}-\sin(x+\pi)$	$3\sqrt[5]{\sin^2 x+2}$
28	$3x^5-\operatorname{ctg}(\pi x^3)$	$(x+1)^{0,3}+\sin 2x^3$	$5x-x^2$
29	$\lvert x\rvert^{\sin(x)}+\sin(x)$	$3^{x+3}+2x$	$2^x+\sin(\pi x)$
30	$x^2+\sin(7x)$	$\lvert x^3+10^x\rvert$	$\sqrt[7]{2x^4+x^2+1}$

Контрольные вопросы

1. Как организуются множественные действия в операторе условия `if`?
2. Какой формат записи имеет тернарный оператор условия?
3. Какой оператор условия рекомендуется использовать для программирования меню?
4. В чем различие и сходство между операторами break и continue?
5. Как можно обеспечить выход из вложенных циклов?
6. Как можно организовать переходы в различные точки программы на **C++**?
7. Какие логические операторы отношения используются в языке **C++**?
8. Что произойдет, если в операторе switch после метки **case** не использовать оператор break?
9. Что произойдет, если в операторе switch не поставить метку default и условие переключения не совпадет ни с одной меткой **case**?

ЛАБОРАТОРНАЯ РАБОТА 4

Организация циклов в языке C++

Теоретическая часть

Операторы цикла относятся к управляющим конструкциям всякого языка программирования. Управляющие операторы и конструкции языка задают порядок, в котором выполняются вычислительные операции программы.

4.1. Оператор `while`

Изучение операторов цикла начнем с оператора while. Цикл while имеет следующий формат (синтаксис) записи:

```
while (expression)
program statement;
```

Производится расчет выражения expression, заключенного в круглые скобки. Если в результате расчета выражения expression получается истинный результат (TRUE), то выполняется утверждение program statement, следующее непосредственно за закрывающей круглой скобкой. После выполнения этого утверждения вновь рассчитывается выражение expression. Если в результате расчета будет TRUE, то вновь будут выполнены утверждения program statement. Цикл повторяется до тех пор, пока в результате расчета выражения expression (в круглых скобках оператора while) не будет получено значение FALSE (ложный), которое является признаком окончания цикла, после чего выполнение программы продолжается с утверждения, следующего за утверждением program statement. Когда требуется выполнить группу утверждений, то она (группа) располагается в фигурных скобках:

```
while (expression)
{
program statement;
program2 statement2;
program3 statement3;
...
}
```

Открывающаяся фигурная скобка может следовать непосредственно после закрывающей круглой скобки оператора while. Все, что находится в фигурных скобках, будет выполняться, пока верно выражение expression.

Очевидно, что неверное задание выражения expression может привести к бесконечному циклу (к зацикливанию).

4.2. Оператор `for`

Оператор цикла for имеет следующий формат записи:

```
for (init_expression; loop_condition; loop_expression)
program statement;
```

Три выражения, заключенные в круглые скобки оператора цикла for, задают условия выполнения программного цикла.

Первый параметр init_expression используется для задания начального значения цикла.

Второй компонент loop_condition определяет условие или условия, в соответствии с которыми будет происходить выход из цикла. Повторение будет происходить до тех пор, пока это условие (или условия) выполняются. Если условие не выполняется, то цикл немедленно заканчивается.

Третий параметр loop_expression выполняется каждый раз, когда заканчивается обработка тела цикла, т.е. program statement.

Чаще всего выражения init_expression и loop_expression являются операторами присваивания или вызовами функций, а второе выражение loop_condition – выражением отношения или логическим выражением. Любую из трех частей можно опустить, но точки с запятыми должны остаться на своих местах. Если опустить init_expression или loop_expression, то соответствующие операции не будут выполняться. Если же опустить проверку условия loop_condition, то по умолчанию считается, что условие продолжения цикла всегда истинно, и тогда цикл станет бесконечным (произойдет зацикливание).

Когда требуется выполнения нескольких утверждений, то они должны заключаться в фигурные скобки:

```
for (init_expression; loop_condition; loop_expression)
{
program1 statement1;
program2 statement2;
program3 statement3;
...
}
```

В представленном случае тело цикла находится в фигурных скобках.

Конструкция цикла, реализованная оператором `for`, может быть выполнена также и оператором `while` следующим образом:

```
init_expression;
while (loop_condition)
{
program statement;
loop_expression;
}
```

Исключением является применение операции continue.

В программах языка **C** возможно применять вложенные циклы, каждый из которых контролируется своей переменной цикла и своим отношением (второе выражение в круглых скобках оператора for). Вложенные циклы могут идти непосредственно друг за другом или составлять тело цикла с помощью фигурных скобок. Возможно также использование двух индексных переменных для инициализации начала цикла с последующим их

инкрементированием (увеличением) или декрементированием (уменьшением).

4.3. Оператор `do-while`

Рассмотренные операторы цикла while и for производят проверку условия выполнения цикла до начала выполнения тела цикла. Поэтому тело цикла может ни разу не выполниться, если с самого начала результатом расчета условия выполнения цикла будет значение FALSE (ложь). В случае необходимости производить проверку условия выполнения цикла после тела цикла (т.е. когда выполняется хотя бы одно предписанное действие в теле цикла) прибегают к циклу do–while.

Оператор цикла do–while имеет следующий формат записи:

```
do
program statement;
while (loop_expression);
```

Выполнение цикла do–while происходит следующим образом: сначала выполняется утверждение program statement, затем производится проверка условия выполнения цикла loop_expression с помощью оператора while. Если результатом проверки будет значение TRUE (истина), то выполнение цикла продолжится, и утверждение program statement всякий раз будет выполняться вновь. Повторение цикла будет продолжаться до тех пор, пока в результате проверки условия выполнения цикла loop_expression будет получаться значение TRUE. Когда в результате проверки условия будет вычислено значение FALSE (ложь), то выполнение цикла прекратится и произойдет переход к утверждению (следующему фрагменту программы), непосредственно следующему за циклом.

Таким образом, цикл do–while гарантированно выполнится хотя бы один раз.

В случае выполнения нескольких утверждений используются фигурные скобки для выделения тела цикла:

```
do {
program1 statement1;
program2 statement2;
program3 statement3;
... } while (loop_expression);
```

Оператор цикла while называется оператором цикла с предусловием, оператор цикла for называется оператором цикла с параметром, оператор цикла do–while называется оператором цикла с постусловием.

Практическая часть

Рассмотрим примеры программ с операторами циклов while, for и do–while.

Пример 1. *Напишите программу вывода на экран пользователя целых положительных чисел с помощью оператора while. Начальное и последнее число должно задаваться пользователем с клавиатуры.*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
int main(void)
 {
int i, j = 0, n;
printf("\n\t Enter the primary natural number: ");
scanf_s("%d", &i);
printf("\t Enter the last natural number: ");
scanf_s("%d", &n);
printf("\n\t The numbers are:");
while (i <= n) {
printf("\n\t %3d", i);
++i;
++j;
}
printf("\n\t Total numbers: %d\n", j);
printf("\n Press any key: ");
      _getch();
      return 0;
 }
```

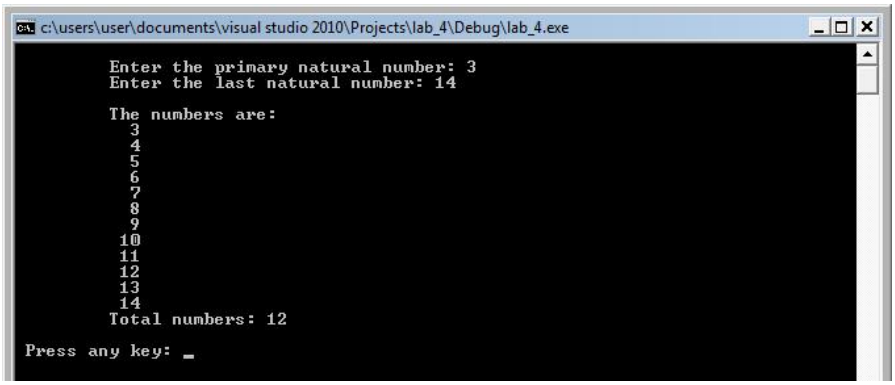


Рис. 4.1. Результат выполнения программы с вводом целых чисел

Возможный результат выполнения программы показан на рис. 4.1.

В программе использована функция scanf_s(), принятая в MS Visual Studio. В программе применено инкрементирование переменных, принятое в языке **C**, а именно ++i или ++j означает, что переменные увеличиваются на единицу. При этом знаки "++" могут располагаться перед именем переменной или после. Отличие в том, что ++i – это значение переменной после увеличения, а i++ – сначала переменная имеет заданное значение, а потом происходит ее увеличение. Для переменных цикла обе формы равнозначны.

Условием цикла является то, что пока переменная i меньше или равна

переменной n (предполагается, что n больше начального значения i), то будут выполняться действия (печать и увеличение переменной j), заложенные в теле цикла. Расчет выражения, заключенного в круглые скобки оператора, предназначен для проверки нестрогого неравенства переменной i по отношению к переменной n. Если это неравенство выполняется, то в теле цикла происходят печать и увеличение (инкрементирование) переменных i, j.

Пример 2. *Напишите программу табличного вывода строчных букв латинского алфавита и их десятичных кодов с помощью оператора цикла for.*

Как известно, в латинском алфавите 26 букв. Поэтому можно создать массив символов этих букв. С учетом того, что тип char представляет собой целочисленный тип, то можно обойтись без создания массива.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
int main(void){
      int j = 1;
      char a = 'a';
      printf("\n  Table code characters:\n");
for ( ; a <= 'z'; ++a)
      printf("\n %4d) %2c: code%4d", j++, a, a);
      printf("\n\n Press any key: ");
      _getch();
      return 0;
}
```

Результат выполнения программы показан на рис. 4.2.

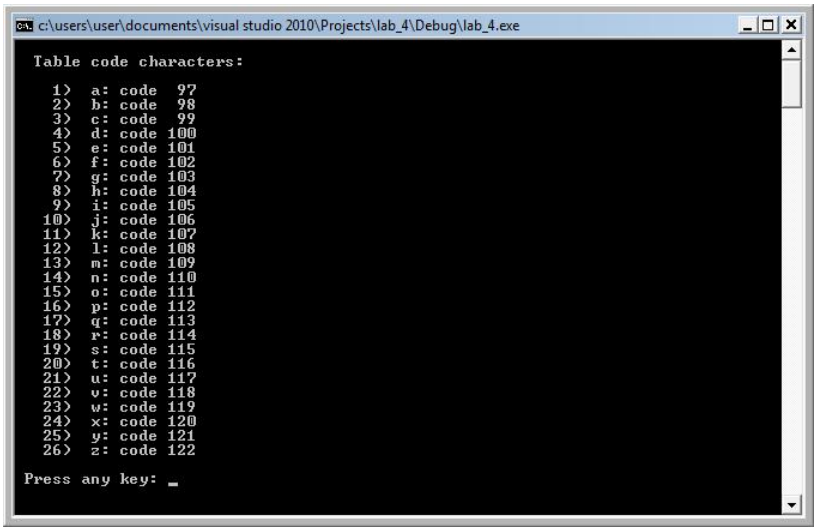


Рис. 4.2. Таблица десятичных кодов букв латинского алфавита

Форматированный вывод данных предусматривает выравнивание по правому краю, для чего предусматриваются числовые спецификаторы типа %4d и %2c для целых чисел и символов в функции printf().

Пример 3. *Вычислите с точностью до "машинного нуля" значение суммы числового ряда:*

$$\frac{1}{1\cdot 2\cdot 3}+\frac{1}{2\cdot 3\cdot 4}+\frac{1}{3\cdot 4\cdot 5}+\frac{1}{4\cdot 5\cdot 6}+\ldots$$

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
int main(void)
 {
   double denom;
   double sum1 = 0.0, sum2 = 0.0;
   int k = 1;
   denom = k * (k + 1) * (k + 2); // знаменатель ряда
      do {
      sum1 = sum2;
      sum2 += 1.0 / denom;
      denom = denom / k * (k + 3);
      ++k;
      } while (sum1 < sum2);
printf("\n\t The amount of numerical series: %lg\n",
sum2);
      printf("\n Press any key: ");
      _getch();
      return 0;
}
```

В приведенной программе сумма вычисляется как значение переменной sum2. Ее предыдущее значение сохраняется в переменной sum1. Так как приближенное значение с добавлением неотрицательных слагаемых не уменьшается, условием продолжения цикла служит отношение sum1 < sum2 (поскольку растет знаменатель denom). Когда при добавлении очередного слагаемого значение суммы остается неизменным (за счет конечной разрядной сетки для представления вещественных чисел), нарушается условие sum1 < sum2 и цикл прекращается. Таким образом, конечность разрядной сетки представления вещественных чисел в компьютере определяет собой "машинный нуль".

Инициализация знаменателя сделана до начала цикла. Форматный вывод результата выполнен с помощью спецификатора символа "l".

Результат выполнения программы показан на рис. 4.3.

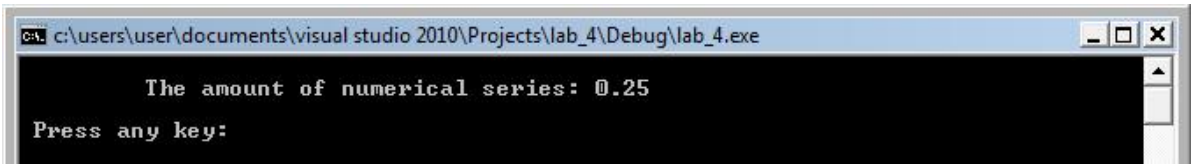


Рис. 4.3. Результат подсчета суммы бесконечного ряда

Пример 4. *Произведите реверс цифр заданного целого числа, вводимого с клавиатуры пользователем.*

Задача заключается в том, чтобы, например, число 123 переписать как 321.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

int main(void)
 {    long int x, r;
      printf("\n Enter an integer: ");
      scanf_s("%ld", &x);
      printf("\n Reverse-digit number %ld:\n\n\t", x);
      do {
      r = x % 10;
      printf(" %ld", r);
      x = x / 10;
      } while ( x != 0);
      printf("\n\n Press any key: ");
      _getch();
      return 0;
}
```

В программе применена операция арифметическая операция деления по модулю, которая имеет символ процента, т.е. "%". Любой остаток, получающийся в результате деления целых чисел, будет отброшен. В шкале старшинства оператор деления по модулю имеет приоритет, равный приоритету операторов умножения и деления. Переменные, используемые в программе, объявлены как длинные числа, поэтому применен тип long int (или long). В некоторых компиляторах имеются отличия между типами int и long int в смысле максимально поддерживаемого значения числа.

Результат выполнения программы показан на рис. 4.4.

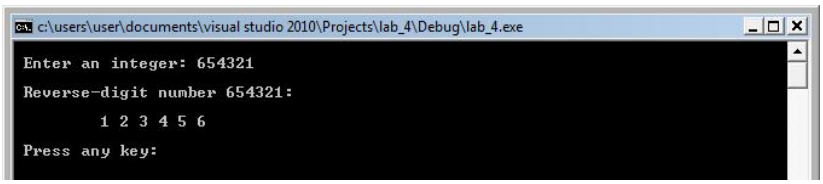


Рис. 4.4. Результат программы по реверсу числа

Пример 5. *На основе только оператора цикла for напишите программу по выводу "горки" заглавных букв, симметрично убывающих к букве, введенной пользователем. Также на основе оператора цикла for предусмотрите защиту от неправильного ввода.*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
```

```
int main (void) {
      int p = 0;
     char ch = 'A';
     char i, j, k, ch2, kk, chA;
     chA = ch;
printf("\n   Enter a capital letter between \"A\" and
\"S\": ");
scanf_s("%c", &ch2, sizeof(char));
for(chA -= 1; chA >= ch2; chA-- )
 {    printf("\n   Error! Press any key: ");
      _getch();
      return -1; }
for (kk = 'S'+1; kk  <= ch2; kk++)
{printf("\n   Error! Press any key: ");
      _getch();
     return -1;}
k = ch2;
for ( kk = ch; kk <=  k; kk++)
 {          printf("\n ");
for (ch2 = ch; ch2 <= k-p ; ch2++)
                 printf("  ");
for (j = ch; j <= kk ; j++)
           printf(" %c", j);
      for (i = kk; i > ch; i-- )
                 printf(" %c", i-1);
      p++;}
    printf("\n\n Press any key: ");
      _getch();
      return 0;
}
```

Пример выполнения программы показан на рис. 4.5.

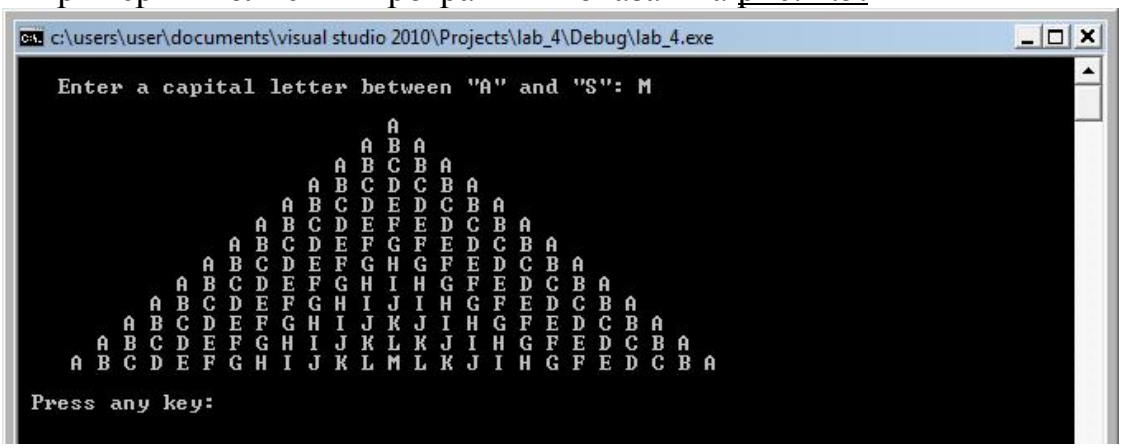


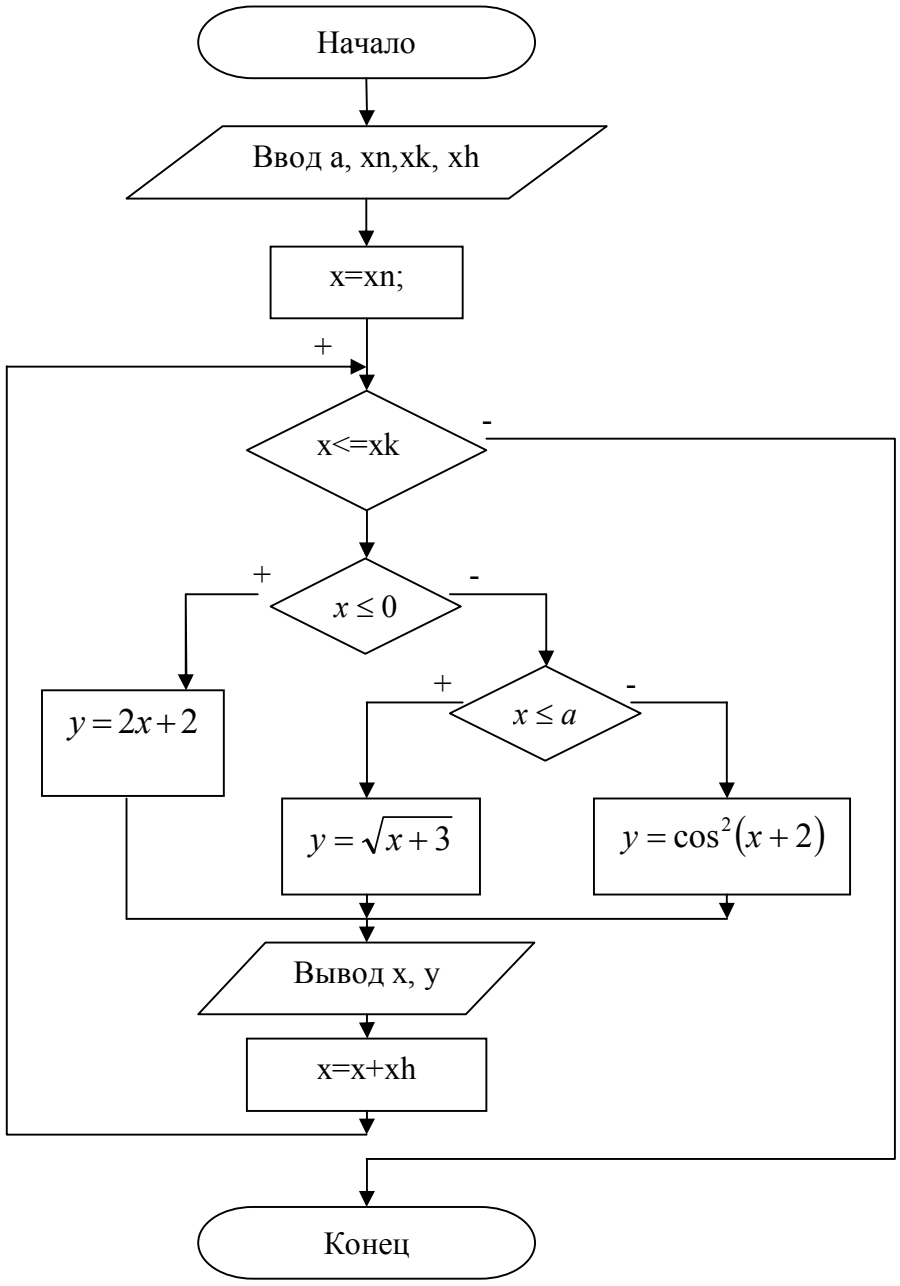
Рис. 4.5. Пример горки букв

Пример 6. *Создать блок-схему к программе и программу на языке программирования СИ++ таблицы табулирования функции*

$$y=\begin{cases}2x+2, & \text{если } x\le 0\\ \sqrt{x+3}, & \text{если } 0<x\le a\\ \cos^2(x+2), & \text{если } x>a\end{cases}$$

с использованием оператора `While` *на отрезке* **[-2; 5]** *с шагом* **0,8.**

Блок-схема к заданию



Программный код решения примера:

```
// Четвертая программа на языке Си++
// Автор Загребельный С.Л.
#include <stdio.h>
#include <conio.h>
```

```
#define _USE_MATH_DEFINES
#include <math.h>
#include <limits.h>
#include <float.h>
int main(void)
 {double xn, xk, xh, a, x, y;
printf("\n\t Vvedite xn= ");
scanf_s("%lf", &xn);
printf("\t vvedite Xk= ");
scanf_s("%lf", &xk);
printf("\t vvedite Xh= ");
scanf_s("%lf", &xh);
printf("\t vvedite a= ");
scanf_s("%lf", &a);
printf("\n\t Tablica znacheniy");
x=xn;
while (x <= xk) {
    if (x<=0){ y=2*x+2;}else {
    if (x<=a) {y=sqrt(x+3);}else{
        if (x>a) {y=pow(cos(x+2),2);}}}
    printf("\n\t %4.3f\t\t%4.3f\n", x,y);
    x=x+xh; }
printf("\n Press any key: ");
    _getch();
    return 0;
 }
```

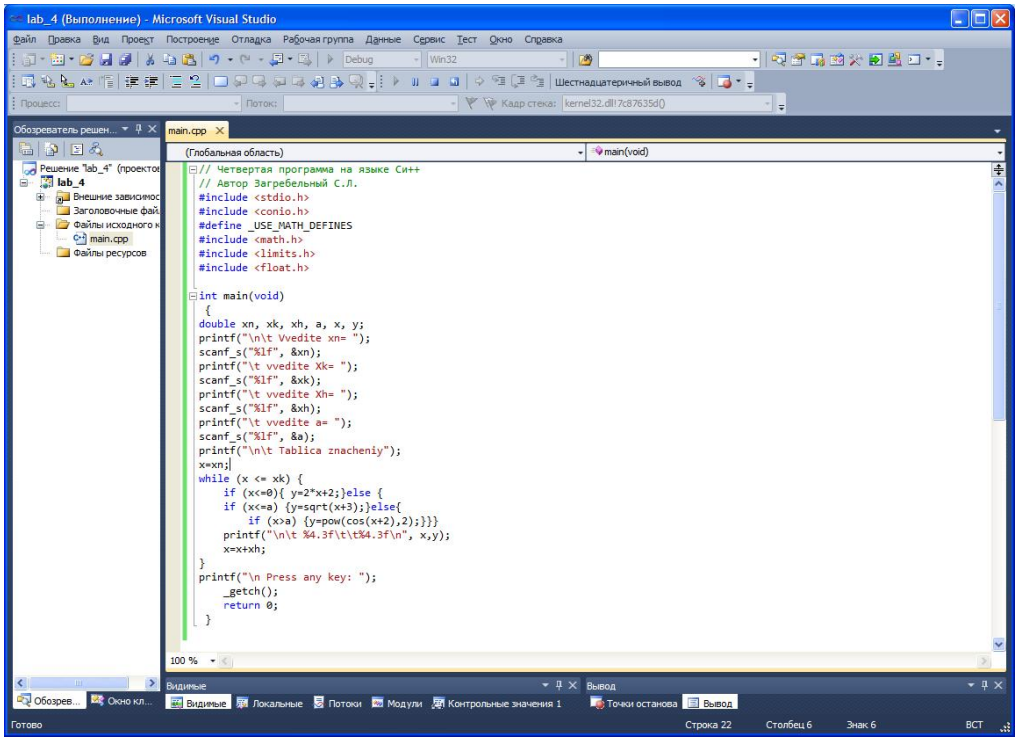


Рис. 4.6. Окно программы

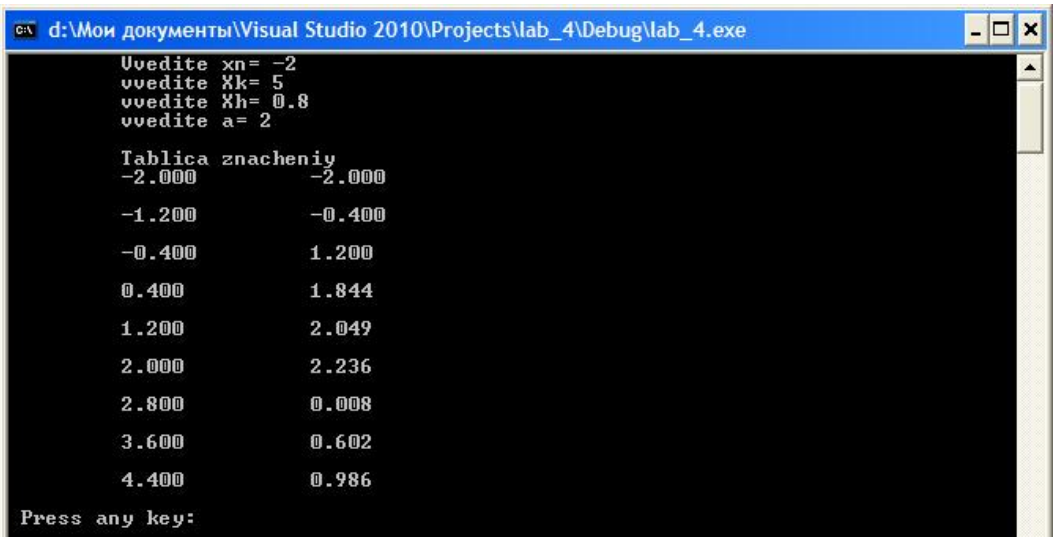


Рис. 4.7. Результат выполнения программы

Индивидуальные задания

Создать блок-схему к программе и программу на языке программирования СИ++ таблицы табулирования функции

$$y=\begin{cases} f1(x), & \text{если } x\le 0 \\ f2(x), & \text{если } 0<x\le a \\ f3(x), & \text{если } x>a \end{cases}$$

с использованием оператора `While` *на отрезке* ***[xn;xk] с шагом xh.*** *Данные взять с таблицы 4.2.*

Таблица 4.2

Вариант	**Функции**			**Границы отрезка**	**Шаг табулирования**
	$f1(x)$	$f2(x)$	$f3(x)$		
1	$\ln\left(x^2+5\right)$	$\sin\left(e^x+2\right)$	$\dfrac{\sin\left(x+3\right)}{e^{2x}+\cos\left(x+1\right)}$	[-2,9; 8,2]	0,2
2	$2\sqrt{\lvert x^3\rvert}\sin\left(x^3\right)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$	[-1,2; 2,6]	0,1
3	$\sin\left(x^5+3\right)$	$\sqrt{x^3}\sin x$	$x^4-\sin\left(x+1\right)$	[-1,7; 2,4]	0,3
4	$x^4\operatorname{tg}\left(x+2\right)$	$\ln\left(4x^2+1\right)$	$\ln\sqrt[5]{5+x^2}$	[-4,3; 8,0]	0,5
5	$x^5+\sqrt[3]{x+10}$	$1{,}3\sqrt{4+x^2}$	$\lvert x+1\rvert^x$	[-9,1; 5,8]	0,14

Продолжение таблицы 4.2.

6	$\|x\|^5 \operatorname{ctg}\|2x\|$	$\ln(x^2+1)$	$e^{-2x}-\sqrt[3]{\|x+1\|}$	[-3,4; 2,5]	0,23
7	$x^5 \operatorname{ctg}(2x^3)$	$\sqrt[5]{x^4+3}$	$\|\sin^2 x+1\|^{2x}$	[-2,2; 8,1]	0,15
8	$\operatorname{ctg}(3x-1)^2$	$2+xe^{-x}$	$\sin(x^3+1)$	[-2,8; 5,2]	0,5
9	$x^3+4x^2\sqrt{\|x\|}$	$(x-1)^3+\cos x^3$	$\sqrt{\|x\|^3}\sin x^3$	[-3,2; 7,8]	0,36
10	$(2x+1)(\|x\|+2)^3$	$e^x+\sin(x+2)$	$3\ln\sqrt[5]{\sin^2 x+2}$	[-6,1; 1,3]	0,15
11	$\operatorname{ctg}(x^3+1)$	$\ln(\sin x+1)^2$	$\sqrt[3]{2x^2+x^4+1}$	[-7,4; 0,6]	0,16
12	$1{,}3\sqrt{4+x^2}$	3^{x+3}	$5^{x+1}+\operatorname{tg}(x+1)$	[-1,2; 7,1]	0,45
13	$e^{2x}+\sin(2x^3)$	$\sin^3 x^4$	$e^{-x}+\sqrt[3]{3x^2+1}$	[-2,2; 3,9]	0,55
14	$x^3+(\|x\|+1)^{0,1x}$	$(x-1)^3+\cos(x^3)$	$2x+\operatorname{tg}(x^2+2)$	[-0,3; 4,5]	0,62
15	$\left\|\sqrt[3]{\frac{2x+5}{(x^3+2)}}\right\|$	$\frac{5x+x^2}{(x^2+3)^3}$	$\cos^2(x^3+\sqrt{x})$	[-2,4; 4,4]	0,4
16	$\sqrt[5]{x^2+x+1}$	$\ln^2(\|\sqrt{x+5}\|)$	$\sin(x^2)+x^{0,25}$	[-3,9; 3,8]	0,15
17	$3x^5+\operatorname{ctg}(x^3+1)$	$e^{x+1}-\sin(\pi x)$	$\sqrt[5]{\sin^2 x+2}$	[-1,3; 7,1]	0,6
18	$x^5-\operatorname{ctg}(\pi x^3)$	$(\|7x\|+1)^{0,3}+\sin x$	$5x-x^2$	[-2,9; 6,2]	0,8
19	$\|x\|^{x+2}+\sin(x)$	$3^{x+3}+2x$	$\sqrt[5]{x^2+x+1}$	[-3,7; 8,5]	0,11
20	$x^2+\sin(7x)-1$	$\|x^3+10^x\|$	$\sqrt[7]{2x^4+x^2+1}$	[-3,9; 1,2]	0,25
21	$\sqrt[3]{\|x\|+2}-1$	$\sin(x^2)+x^{0,25}$	$\ln^2(x)+\sqrt{x}$	[-4,5; 6,1]	0,3
22	$\sqrt{\|\sin^2 x+\cos^4 x\|}$	$\ln(x+1)+\sqrt{3x}$	$5^{x+1}+\operatorname{tg}(x+3)^2$	[-3,4; 3,4]	0,33
23	$x^3-3x^2\sqrt{\|x\|+6}$	$\frac{2x+2}{\operatorname{tg}(2x-1)+1}$	x^4-x^x	[-4,1; 5,0]	0,45
24	x^4+2x^3-x	$e^{-x}+\sqrt[3]{x}$	$\ln(\|x^3+x^2\|)$	[-1,7; 2,9]	0,75
25	$\frac{(3x-1)^2}{x^5+2x+1}$	$\ln^2\|\sqrt{x+5}\|$	$\sqrt[5]{1+x^2}$	[-1,6; 4,7]	0,65

Продолжение таблицы 4.2.

26	$x^5 \operatorname{ctg}(2x^3)$	$\frac{5}{\operatorname{tg}(2x+3)+1}$	$x^2 e^{-x}$	[-1,6;3,7]	0,3
27	$x^x \operatorname{tg}(x+5)$	$x^3 \cos x$	$\sin x^2 + x^{0.25}$	[-2,8; 8,2]	0,4
28	$\ln(x+1)+\sqrt{3x}$	$\sin(x^2+3x)$	$\ln^2\lvert x\rvert+\sqrt{x+3}$	[-1,7; 2,6]	0,25
29	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$\sin(x-e^{-x})$	[-2,2; 7,4]	0,23
30	$\cos(x^3+1)e^{-x}$	$\left\lvert\sqrt[3]{\frac{2x+5}{(x^3+2)}}\right\rvert$	$2\sqrt{x^2+7\sin(x^3)}$	[-1,1; 7,9]	0,8

Контрольные вопросы

1. Как организуются составные операторы циклов в языке **C**?
2. Как организуются вложенные циклы в языке **C**?
3. В каких случаях может произойти зацикливание при использовании оператора цикла с предусловием?
4. В каких случаях может произойти зацикливание при использовании оператора цикла с постусловием?
5. Сколько условий требуется для работы оператора цикла с параметром?
6. Сколько операторов отношения в языке **C**? Перечислите их.
7. Как реализуется взаимозаменяемость операторов цикла while и for?
8. В чем сходство и различие между циклами с предусловием и с постусловием?

ЛАБОРАТОРНАЯ РАБОТА 5

Одномерные числовые массивы в языке программирования C++. Селективная обработка элементов массива. Нахождение минимального и максимального элементов массива.

Теоретическая часть

В языке программирования **C** заложены средства для задания последовательностей упорядоченных данных. Такие последовательности называются массивами. В массивах должны быть упорядочены данные одного и того же типа. В данной лабораторной работе будут рассматриваться массивы с целыми и вещественными типами данных, т.е. типы int, float или double.

Массивы данных могут быть одномерными (векторами размера $1 \times n$ или $n \times 1$), двухмерными (матрицами размера $n \times m$) или многомерными (размера $n \times m \times p$...). В частности, для векторов и матриц в приведенной записи первый индекс означает количество строк, а второй (число или буква) – это количество столбцов. Для названия массива может быть использована переменная, состоящая из букв (буквы), букв с цифрами, букв с цифрами и знаком подчеркивания и т.д. в соответствии с правилами объявления переменных, принятых в языке **C**. Если размерность массива меньше, чем требуется, то компилятор не выдаст сообщения об ошибке. Выход за границы массивов должен следить только сам программист.

5.1. Одномерные массивы

Одномерный массив – это список связанных однотипных переменных.

Общая форма записи одномерного массива:

```
тип  имя_массива[размер];
```

В приведенной записи элемент тип объявляет базовый тип массива. Количество элементов, которые будут храниться в массиве с именем имя_массива, определяется элементом размер.

В языке **C** индексация массива начинается с нуля. Например, если размер массива определен величиной 9, то в массиве можно хранить 10 элементов с индексацией 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Доступ к отдельному элементу массива осуществляется с помощью индекса. Индекс описывает позицию элемента внутри массива.

Все массивы занимают смежные ячейки памяти, т.е. элементы массива в памяти расположены последовательно друг за другом. Ячейка памяти с наименьшим адресом относится к первому элементу массива, а с наибольшим – к последнему.

Для одномерных массивов общий размер массива в байтах вычисляется по формуле:

```
всего байт = размер типа в байтах * количество элементов
```

В языке **C** нельзя присвоить один массив другому. Для передачи

элементов одного массива другому необходимо выполнить присвоение поэлементно.

5.2. Инициализация массива

В языке **C++** массив при объявлении можно инициализировать.

Общая форма инициализации массива:

```
тип  имя_массива[размер1] * [размерN] =
{список_значений};
```

В список_значений входят констант, разделенных запятыми. Типы констант должны быть совместимыми с типом массива.

Пример инициализации одномерного массива:

```
int A[5] = {1, 2, 3, 4, 5};
```

При этом A[0] = 1, A[1] = 2 и т.д.

В языке **C** возможна инициализация безразмерных массивов. Например, для одномерного массива:

```
int A[ ] = {1, 2, 3, 4, 5};
```

Практическая часть

Пример 1. *Напишите программу заполнения одномерного массива случайными числами из интервала от 1 до 15 по случайному равномерному закону. Отсортировать массив случайных чисел по возрастанию.*

Для решения поставленной задачи применим сортировку методом прямого выбора. Алгоритм сортировки заключается в следующем:

1. В исходной последовательности из N элементов отыскивается элемент с наименьшим ключом.
2. Он меняется местами с первым элементом.
3. В оставшейся последовательности из (N–1) элементов отыскивается минимальный элемент и меняется местами со вторым элементом и т.д., пока не останется один, самый большой элемент.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
#define Left 1
#define Right 15
#define N 10

int main (void) {
float R, r, min;
float A[N];

int i, j, k;
unsigned int some;
```

```
long int L;

L = (long) time(NULL); // Системное время
some = (unsigned) L; // Приведение типов
srand(some); // Задание исходного случайного числа для
rand()
printf("\n\t The initial array of random numbers in the
interval [%d, %2d]\n", Left, Right);

for (i = 0; i < N; ++i)
{// Случайное число из интервала [0,1]
r = (float) rand()/RAND_MAX;
// Формирование случайного числа из заданного интервала
R = Left + (Right - Left) * r;
// Заполнение массива случайными числами
A[i] = R; }

// Печать элементов исходного массива
for (i = 0; i < N; ++i)
printf("\n\t %5d) %10.4f", i + 1, A[i]);

// Сортировка методом выбора
for (i = 0; i < (N - 1); ++i)
 {
 min = A[i]; k = i;
for (j = i + 1; j < N; ++j)
if (A[j] < min) { k = j; min = A[k]; }
A[k] = A[i]; A[i] = min;
}

// Печать отсортированного массива по возрастанию
printf("\n\n\t Sort an array:\n");
for (i = 0; i < N; ++i)
printf("\n\t %5d) %10.4f", i + 1, A[i]);

      printf("\n\n Press any key: ");
      _getch();
      return 0;
}
```

Возможный результат выполнения программы показан на рис. 5.1.

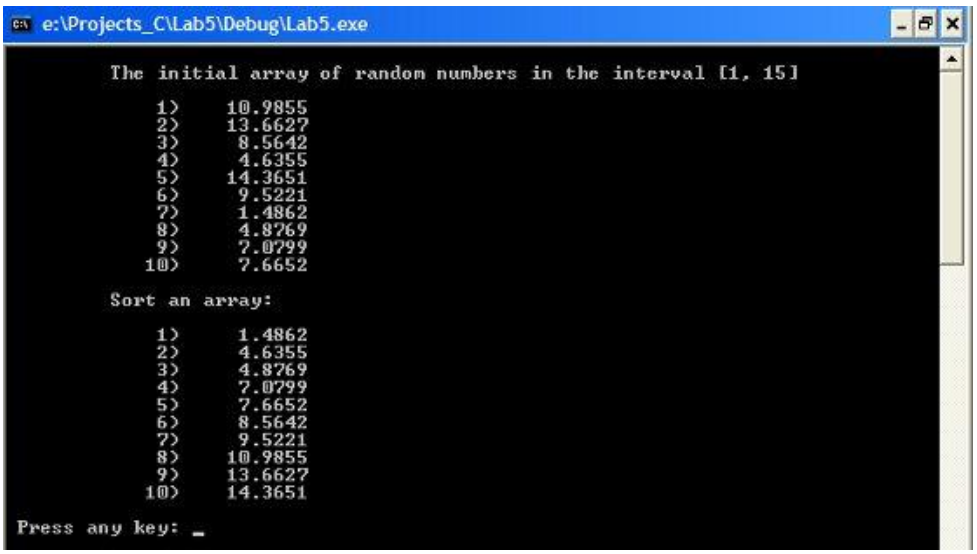


Рис. 5.1. Сортировка одномерного массива по возрастанию

В программе использованы директивы препроцессора для задания левой границы (#define Left 1), правой границы (#define Right 15) и размера одномерного массива (#define N 10). Включены дополнительные библиотеки **time.h** – для обращения к функциям системного времени, **stdlib.h** – для обращения к функциям генерации псевдослучайных чисел.

Пример 2. *Напишите программу поиска максимального элемента в заданном одномерном массиве. Элементы массива являются целыми числами.*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

int main (void)
{
      int i, size, max;
      int A[ ] = {3, 5, 2, 8, 12, 0, -7, -3, -21};
size = sizeof(A)/sizeof(A[0]);
printf("\n\t The dimention of the array A is equal to:
%d\n", size);

max = A[0]; // Предполагаемый максимум
      for (i = 0; i < size; ++i)
            if (A[i] > max) max = A[i];
printf("\n\t Maximum array element: %d\n", max);

printf("\n\n Press any key: ");
      _getch();
      return 0;
}
```

В программе использована инициализация безразмерного массива и определения его размерности с помощью функции sizeof().

Результат выполнения программы показан на рис. 5.2.

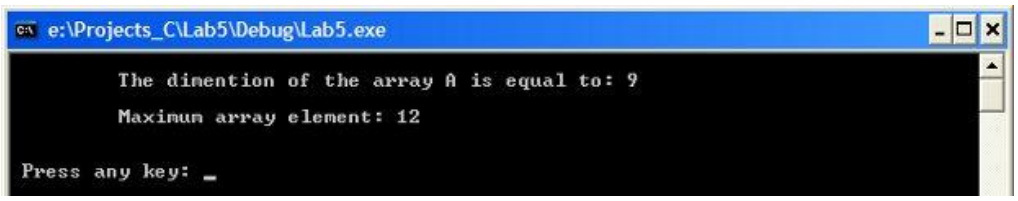


Рис. 5.2. Определение максимального элемента массива

Пример 3. *Напишите программу циклической перестановки чисел заданного массива так, чтобы i-е число стало (i+1)-м, а последнее число – первым. Выведите на дисплей исходный массив и преобразованный.*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#define N 55

int main (void)

{
      int i, j, k;
      double D[ ] = {1.23, 2.34, 3.45, 4.56, 5.67,
6.78};
      double B[N];
// Заведомо больший размер, чем у массива D

// Обнуление массива и  выделение памяти для него
      for (i = 0; i < N; ++i)
            B[i] = 0.0;

k = sizeof(D)/sizeof(D[0]);

B[0] = D[k-1];
for (i = 0; i < (k - 1); ++i)
B[i+1] = D[i];

printf("\n\t The original array:\n");
for (i = 0; i < k; ++i)
printf("%8.2f", D[i]);

printf("\n\n\t The reconfigured array:\n");
for (j = 0; j < k; ++j)
printf("%8.2f", B[j]);
```

```
    printf("\n\n Press any key: ");
    _getch();
    return 0;
}
```

Результат выполнения программы показан на рис. 5.3.

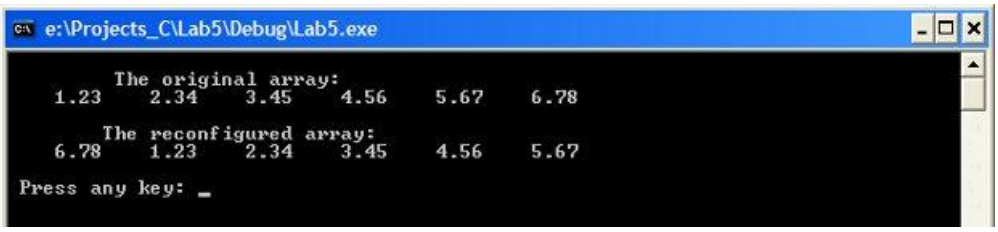


Рис. 5.3. Пример циклической перестановки элементов числового массива

Пример 4. *В данном одномерном массиве вещественных чисел поменяйте местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах. Предусмотрите четность и нечетность размерности массива.*

Для определения четности места в заданном массиве можно использовать операцию деления по модулю, т.е. **%**.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

// Размер массива
#define n 7
int main (void) {
int i, k;

// Пример массива
float A[n] = {1.23F, 2.34F, 3.45F, 4.56F, 5.67F, 6.78F,
7.89F};
float B[n];// Вспомогательный массив

// Обнуление массива
     for (i = 0; i < n; ++i)
          B[i] = 0;
// Распечатка заданного массива
printf("\n\t\t The original array of dimention n =
%d:\n", n);

printf("\t");
```

```
      for (i = 0; i < n; ++i)
           printf("%6.2f", A[i]);

// Распечатка преобразованного массива
     printf("\n\n\t\t The reconfigured array:\n");
     for (i = 0; i < n; ++i) {

k = i % 2; // Для определения четности индекса массива
if (k == 0 && i < n - 1 )
 B[i] = A[i + 1];
else if (k != 0 && i > 0 )
B[i] = A[i-1];
else if (k == 0 && i < n)
B[i] = A[i]; }

printf("\t");
          for (i = 0; i < n; ++i)
           printf("%6.2f", B[i]);

printf("\n\n Press any key: ");
     _getch();
     return 0; }
```

При инициализации массива каждый его элемент снабжен суффиксом **F**.

Результат выполнения программы показан на рис. 5.4.

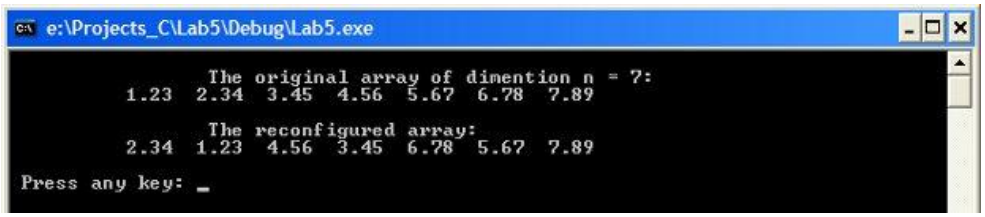
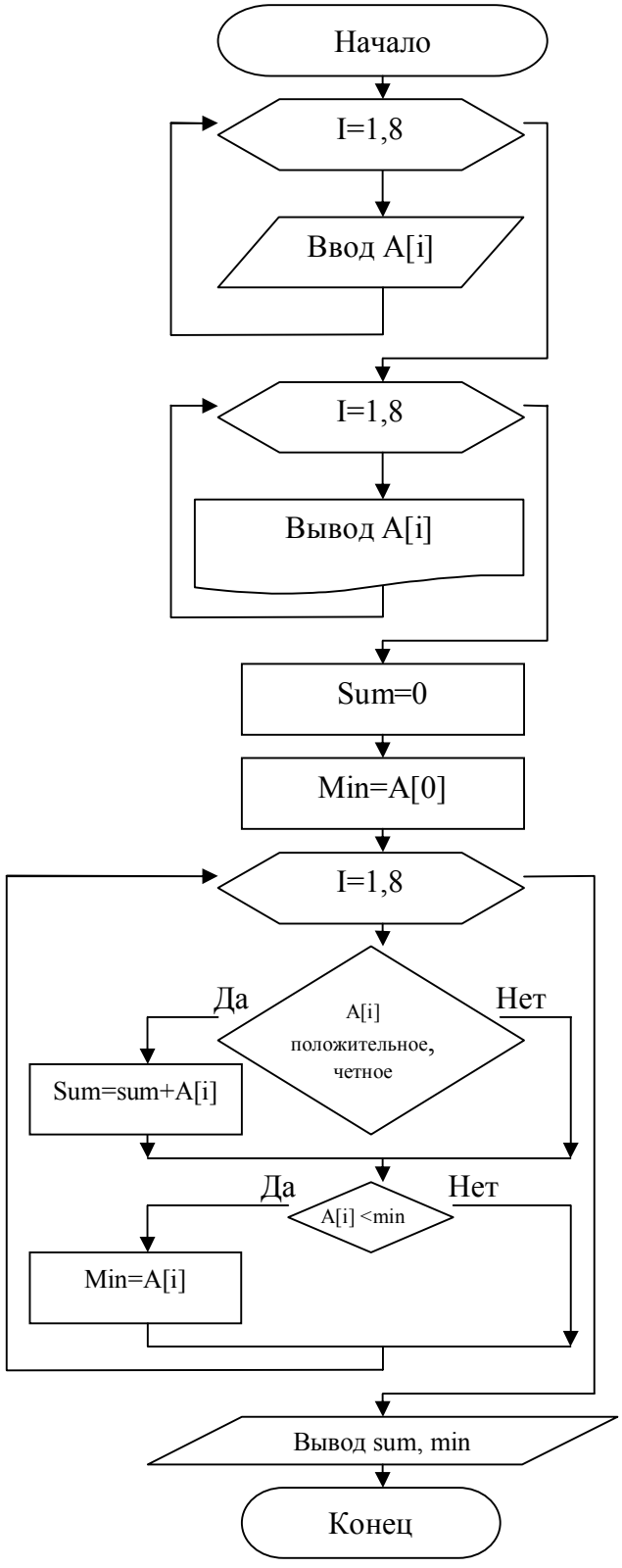


Рис. 5.4. Смена четных и нечетных мест чисел массива

Пример 5. *Создать блок-схему к заданию и программу на СИ++, если дан массив A(8), элементы которого нужно ввести с клавиатуры, найти сумму четных положительных элементов и минимальный элемент массива.*

Блок-схема к заданию



Программный код решения примера:

```
// Пятая программа на языке Си++
// Автор Загребельный С.Л.
#include <stdio.h>
```

```
#include <conio.h>
#include <math.h>

int main (void)
{
int A[8];
int i, sum,min;
printf("\n\t Vvedite massiv iz 8 chisel\n");
for (i = 0; i < 8; i++)
scanf("%i",&A[i]);
// Печать элементов исходного массива
printf("\n\t ishodniy massiv \n");
for (int i = 0; i < 8; i++)
printf("\n\t %5d element massiva %10d", i + 1, A[i]);
// Нахождение суммы четных положительных элементов
sum=0;min=A[0];
for (i = 0; i < 8; i++)
    { if((A[i]>0)&&(!(A[i]%2))) {sum=sum+A[i];}
      if (A[i] < min) {min = A[i];}}
printf("\n\t Minimalniy elementmassiva %d\n", min);
// Печать найденной суммы
printf("\n\n\t Summa=:%10d\n",sum);
    printf("\n\n Press any key: ");
    _getch();
    return 0;
}
```

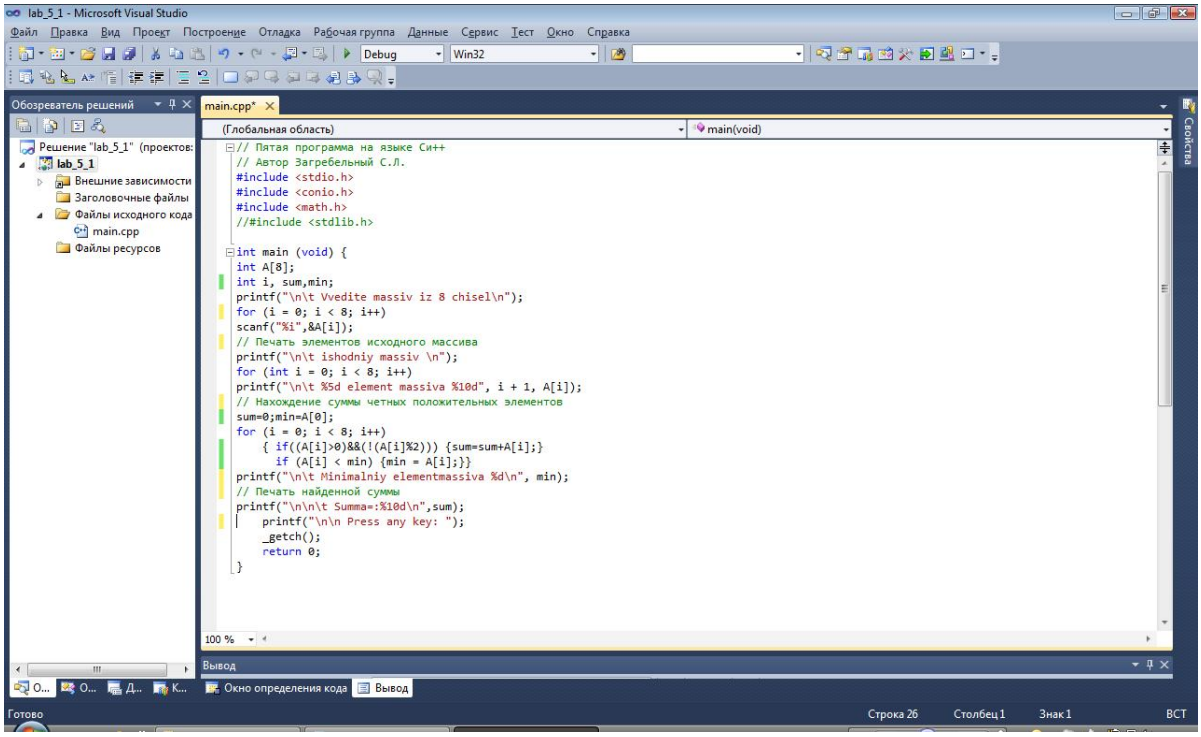


Рис. 5.5 Окно кода программы

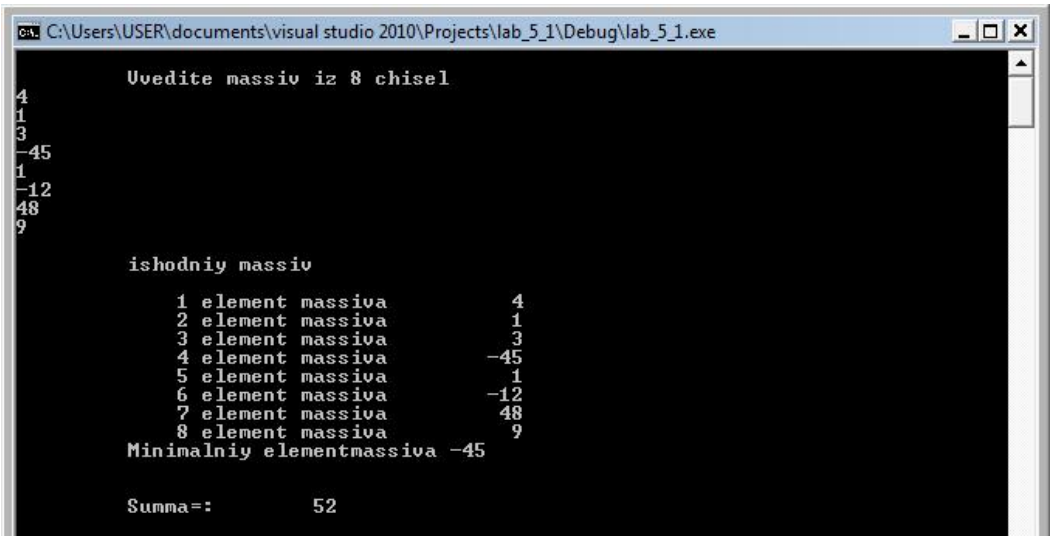


Рис. 5.6. Окно обработки элементов массива

Индивидуальные задания

Составить блок-схемы и программы для решения следующих задач таблица 5.1..

Таблица 5.1.

Вариант	задание	Условие задания
1	***а***	Найти количество положительных элементов
	б	Найти сумму элементов больших 3
	в	Найти максимальный элемент массива
2	***а***	Найти количество отрицательных элементов
	б	Найти сумму отрицательных элементов
	в	Найти минимальный элемент кратный пяти
3	***а***	Найти количество четных элементов
	б	Найти сумму элементов кратных 3
	в	Найти разность максимального и минимального элементов массива
4	***а***	Найти среднее арифметичекое элементов массива
	б	Найти сумму наибольшего и наименьшего элементов массива
	в	Найти максимальный по модулю элемент массива
5	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму минимального положительного элемента массива и его номера
	в	Найти произведение модулей наибольшего отрицательного и наименьшего четного элементов массива

Продолжение таблицы 5.1.

6	***а***	Найти количество элементов кратных 5
	б	Найти сумму четных элементов массива стоящих на нечетных местах
	в	Найти сумму второго и наибольшего положительного элементов массива
7	***а***	Найти среднее геометрическое четных элементов массива
	б	Найти номер наибольшего по модулю элемента массива
	в	Найти максимальный четный элемент массива
8	***а***	Вычислить среднее арифметическое максимального и минимального элементов массива
	б	Найти минимальный по модулю элемент массива
	в	Найти сумму элементов из интервала [0;10]
9	***а***	Вычислить среднее геометрическое номеров максимального и минимального элементов массива
	б	Найти разность суммы положительных и произведения отрицательных чисел массива
	в	Найти количество положительных элементов
10	***а***	Найти сумму отрицательных элементов
	б	Найти сумму элементов массива, у которых индекс кратен 3
	в	Найти произведение модулей наибольшего и наименьшего элементов массива
11	***а***	Найти сумму отрицательных элементов
	б	Найти сумму второго и наибольшего положительного элементов массива
	в	Найти разность максимального и минимального элементов массива
12	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму элементов в диапазоне [-10;20]
	в	Найти максимальный по модулю элемент массива
13	***а***	Найти сумму минимального положительного элемента массива и его номера
	б	Найти сумму четных элементов массива из диапазона [-20;30]
	в	Найти минимальный по модулю элемент массива

Продолжение таблицы 5.1.

14	***а***	Найти количество элементов кратных 5
	б	Найти сумму четных элементов массива стоящих на нечетных местах
	в	Найти максимальный четный элемент массива
15	***а***	Найти количество положительных элементов
	б	Найти сумму элементов больших 3
	в	Найти максимальный элемент массива
16	***а***	Найти количество отрицательных элементов
	б	Найти сумму отрицательных элементов
	в	Найти минимальный элемент кратный пяти
17	***а***	Найти количество четных элементов
	б	Найти сумму элементов кратных 3
	в	Найти разность максимального и минимального элементов массива
18	***а***	Найти среднее арифметичекое элементов массива
	б	Найти сумму наибольшего и наименьшего элементов массива
	в	Найти максимальный по модулю элемент массива
19	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму минимального положительного элемента массива и его номера
	в	Найти произведение модулей наибольшего отрицательного и наименьшего четного элементов массива
20	***а***	Найти количество элементов кратных 5
	б	Найти сумму четных элементов массива стоящих на нечетных местах
	в	Найти сумму второго и наибольшего положительного элементов массива
21	***а***	Найти среднее геометрическое четных элементов массива
	б	Найти номер наибольшего по модулю элемента массива
	в	Найти максимальный четный элемент массива
22	***а***	Вычислить среднее арифметическое максимального и минимального элементов массива
	б	Найти минимальный по модулю элемент массива
	в	Найти сумму элементов из интервала [0;10]

Продолжение таблицы 5.1.

23	***а***	Вычислить среднее геометрическое номеров максимального и минимального элементов массива
	б	Найти разность суммы положительных и произведения отрицательных чисел массива
	в	Найти количество положительных элементов
24	***а***	Найти сумму отрицательных элементов
	б	Найти сумму элементов массива, у которых индекс кратен 3
	в	Найти произведение модулей наибольшего и наименьшего элементов массива
25	***а***	Найти сумму отрицательных элементов
	б	Найти сумму второго и наибольшего отрицательного элементов массива
	в	Найти разность максимального и минимального элементов массива
26	***а***	Вычислить среднее арифметическое четных элементов массива
	б	Найти сумму элементов в диапазоне [-10;20]
	в	Найти максимальный по модулю элемент массива
27	***а***	Найти сумму минимального положительного элемента массива и его номера
	б	Найти сумму четных элементов массива из диапазона [-20;30]
	в	Найти минимальный по модулю элемент массива
28	***а***	Максимальный по модулю элемент
	б	Найти среднее арифметичекое элементов массива
	в	Найти сумму отрицательных элементов
29	***а***	Найти количество элементов кратных 4
	б	Найти сумму отрицательных элементов
	в	Найти сумму наибольшего и наименьшего элементов массива
30	***а***	Найти разность максимального и минимального положительных элементов
	б	Найти сумму нечетных элементов
	в	Найти минимальный элемент из диапазона [-20;30]

Контрольные вопросы

1. Как организуются одномерные числовые массивы в языке **C++**?
2. Как организуется индексирование числовых массивов в языке **C++**?

3. На кого или на что возлагается контроль границ числовых массивов в языке программирования **C++**?

4. Для чего применяется начальная инициализация числовых массивов при дальнейшем их использовании?

5. Почему при определении размерности массива с помощью препроцессорной директивы **define** не используется точка с запятой после числового значения?

ЛАБОРАТОРНАЯ РАБОТА 6

Понятие многомерного массива. Обработка элементов матриц.

Теоретическая часть

6.1. Двухмерные массивы, матрицы

Двухмерный массив представляет собой список одномерных массивов.

Общая форма записи двухмерного массива:

```
тип  имя_массива[размер1] [размер2];
```

В приведенной записи размер1 означает количество строк двухмерного массива, а размер2 – количество столбцов.

В двухмерном массиве позиция любого элемента определяется двумя индексами.Индексы каждого из размеров массива начинаются с 0 (с нуля).

Место хранения для всех элементов массива определяется во время компиляции. Память, выделенная для хранения массива, используется в течение всего времени существования массива.

Для двухмерных массивов общий размер массива в байтах вычисляется по формуле:

```
всего байт = число строк * число столбцов * размер типа в байтах
```

6.2. Многомерные массивы

Общая форма записи многомерного массива:

```
тип  имя_массива[размер1] [размер2]... [размерN];
```

Индексация каждого размера начинается с нуля. Элементы многомерного массива располагаются в памяти в порядке возрастания самого правого индекса. Поэтому правый индекс будет изменяться быстрее, чем левый (левые).

При обращении к многомерным массивам компьютер много времени затрачивает на вычисление адреса, так как при этом приходится учитывать значение каждого индекса. Следовательно, доступ к элементам многомерного массива происходит значительно медленнее, чем к элементам одномерного. В этой связи использование многомерных массивов встречается значительно реже, чем одномерных или двухмерных массивов.

Для многомерных массивов общий размер многомерного массива в байтах вычисляется по формуле:

```
всего байт = размер1* размер2*...* размерN *размер типа в байтах
```

Очевидно, многомерные массивы способны занять большой объем памяти, а программа, которая их использует, может очень быстро столкнуться с проблемой нехватки памяти.

Для определения размера типа в байтах применяется функция `sizeof()`, которая возвращает целое число. Например, sizeof(float).

6.3. Инициализация массивов

При инициализации многомерного массива для улучшения наглядности элементы инициализации каждого измерения можно заключать в фигурные скобки.

Пример инициализации двухмерного массива:

```
int  MN[3][4] = {
{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12}
};
```

Массив MN[3][4] – это матрица, у которой 3 строки и 4 столбца.

Для многомерных массивов инициализацию можно также проводить с указанием номера инициализируемого элемента.

Пример инициализации трехмерного массива:

```
int  XYZ[2][3][4] = {
{ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} },
{ {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23,
24} }
};
```

Как видно, массив XYZ содержит два блока, каждый из которых есть матрица размера 3 × 4, т.е. 3 строки и 4 столбца.

В многомерном массиве размер самого левого измерения также можно не указывать. В частности, для инициализации массива MN[3][4] допустима следующая запись:

```
int  MN[][4] = {
{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12}
};
```

При инициализации многомерных массивов необходимо указать все данные (размерности) за исключением крайней слева размерности. Это нужно для того, чтобы компилятор смог определить длину подмассивов, составляющих массив, и смог выделить необходимую память. Рассмотрим пример безразмерной инициализации для трехмерного массива целых чисел:

```
int  XYZ[][3][4] = {
{
{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12}
},
{
{13, 14, 15, 16},
```

```
    {17, 18, 19, 20},
    {21, 22, 23, 24}
    }
    };
```

Вывод трехмерного массива на консоль (дисплей) можно выполнить по следующей программе:

```
#include <stdio.h>
#include <conio.h>

int main (void) {
     int i, j, k;
     int  XYZ[][3][4] = {
{ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} }, // 1-й
{ {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}
} }; // 2-й
     for (i = 0; i < 2; ++i) { printf("\n");
          for (j = 0; j < 3; ++j) { printf("\n");
               for (k = 0; k < 4; ++k)
                    printf(" %3d", XYZ[i][j][k]);
          }
     }
     printf("\n\n Press any key: ");
     _getch();
     return 0;
 }
```

Практическая часть.

Пример 1. *Напишите программу заполнения квадратной матрицы (заданного размера n > 2) по спирали натуральными числами начиная с левого верхнего угла (принимая его за номер 1) и двигаясь по часовой стрелке.*

Образец заполнения:

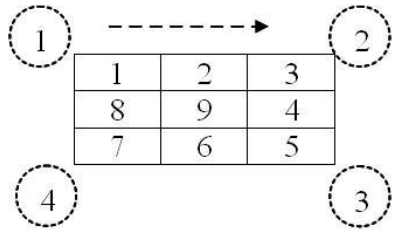


Рис.6.1. Образец заполнения матрицы числами по спирали

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#define n 13
```

```
int main(void) {
      int i = 1, j, k;
      int p = n/2;
      int A[n][n];

      // Обнуление матрицы
      for (j = 0; j < n; ++j)
            for (k = 0; k < n; ++k)
                  A[j][k] = 0;

printf("\n\t Spiral matrix of dimention (%d x %d):\n",
n, n);

      for (k = 1; k <= p; k++)  // Число спиралей
      {
            // Верхний горизонтальный столбец
            for (j = (k-1); j < (n-k+1); j++)
                  A[(k-1)][j] = i++;

            // Правый верхний столбец
            for (j = k; j < (n-k+1); j++)
                  A[j][n-k] = i++;

            // Нижний горизонтальный столбец
            for (j = (n-k-1); j >= (k-1); --j)
                  A[n-k][j] = i++;

            // Левый верхний столбец
            for (j = (n-k-1); j >= k; j--)
                  A[j][(k-1)] = i++;
      }
      if ( n % 2 )
            A[p][p] = n*n;

      // Распечатка матрицы
      for (i = 0; i < n; ++i)
            for (j = 0; j < n; ++j)
            {
                  printf("%5d", A[i][j]);
                  if (j == (n-1))
                        printf("\n");
            }

                        printf("\n Press any key: ");
            _getch();
```

```
        return 0;
}
```

Результат выполнения программы показан на рис. 6.2.

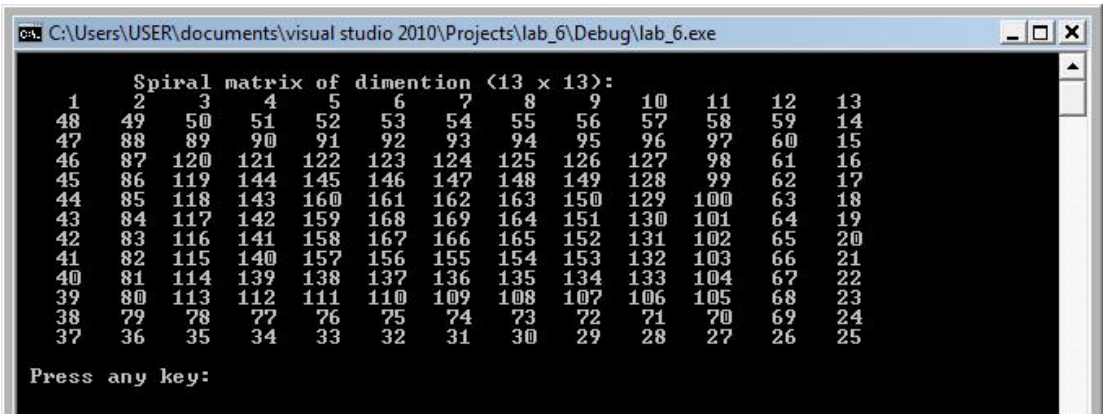


Рис. 6.2. Заполнение матрицы по спирали

Пример 2. *Каждый день производятся замеры некоторых величин (вещественных значений), причем значения этих величин сводятся в прямоугольную таблицу размера n × m. Составьте многомерный массив данных за 30 дней. Формирование данных произвести по случайному равномерному закону из интервала от –12 до 21.*

Этот пример относится к определению трехмерного массива данных.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
#define n 6
#define m 7
#define N 30
const int Left = -12; // Левая граница
const int Right = 21; // Правая граница

int main (void)
{
 float R, r;
 float A[N][n][m];

 int i, j, k;

 // Инициализация генератора случайных чисел
 srand((unsigned) time(NULL));
```

```
      printf("\n\t The values of every 10 days from 30
days:");

      // Формирование данных за 30 дней
      for (k = 0; k < N; ++k)
            for (i = 0; i < n; ++i)
                  for (j = 0; j < m; ++j)
                  {     r = (float) rand()/RAND_MAX;
      R = Left + (Right - Left)*r;
      A[k][i][j] = R;
      }

      // Печать данных за каждый 10-й день
      for (k = 0; k < N; k += 10) { printf("\n");
      for (i = 0; i < n; ++i) { printf("\n");
      for (j = 0; j < m; ++j)
            printf("%10.4f", A[k][i][j]);
      }}

      printf("\n Press any key: ");
      _getch();
      return 0;
    }
```

В программе используется трехмерный массив размера 30 × 6 × 7. Это означает, что прямоугольная таблица (массив) данных размера 6 × 7 как бы скрепляется 30 раз – по заданному числу дней. Границы случайных чисел определены с помощью спецификатора **const**.

Возможный результат выполнения программы показан на рис. 6.3.

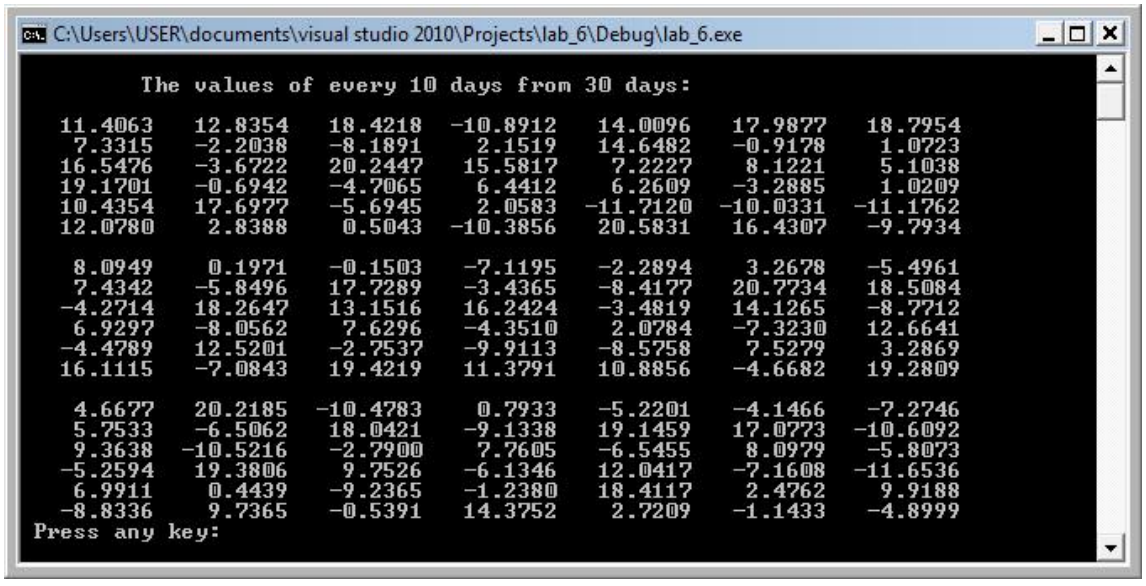


Рис. 6.3. Вывод данных за каждый 10-й день

Пример 3. *Напишите программу по перемножению двух матриц A и B с размерностями (m × r) и (r × n) соответственно. Матрицу A примите размером 4 × 5, матрицу B – размером 5 × 3 (обе целочисленные).*

Условием перемножения двух матриц A и B является равенство числа столбцов матрицы A и числа строк матрицы B. Если первая матрица A имеет размер m × r, то вторая матрица B должна иметь размер r × n. В результате перемножения получим матрицу C размера m × n. Приведем следующую схему по размерностям:

$$C = A * B = m \times r * r \times n = m \times n$$

Поэлементное перемножение двух матриц в стандартной математической форме имеет следующий вид:

$$C_{ij} = \sum_{k=1}^{n} a_{il} \cdot b_{kj}, \quad i = 1, 2, 3..., n, \qquad j = 1, 2, 3..., m$$

С учетом синтаксиса формирования массивов в языке C индексация должна начинаться с нуля, поэтому формулу перепишем в следующем виде:

$$C_{ij} = \sum_{k=1}^{n-1} a_{il} \cdot b_{kj}, \quad i = 1, 2, 3..., n-1, \qquad j = 0, 1, 2, 3..., m-1.$$

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

#define m 4
#define r 5
#define n 3

int main (void) {
 int i, j, k; // переменные циклов

 const int A[m][r] = {{1,2,3,4,5},
 {2,3,4,5,6},
 {2,2,2,2,2},
 {3,3,3,3,3}};

 const int B[r][n] = {{9,8,7},
 {1,2,3},
 {4,5,6},
 {7,8,9},
 {1,1,1}};

 // Массив под результат произведения двух матриц
 int C[m][n];
```

```
      // Обнуление результирующей матрицы
      for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                  C[i][j] = 0;
     // Формирование результата произведения двух матриц
      for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                  for (k = 0; k < r; k++)
                        C[i][j] = C[i][j] +
A[i][k]*B[k][j];
     // Распечатка результата произведения двух матриц
      printf("\n 1) Index: \"ijk\".  Matrix (%dx%d):\n",
m, n);
      for (i = 0; i < m; i++) {
            printf("\n");
            for (j = 0; j < n; j++)
                  printf(" %4d", C[i][j]);
      }
      printf("\n\n ... Press any key: ");
      _getch();
      return 0;
     }
```

В программе используются три цикла по формированию произведения двух матриц. Первый цикл (переменная i) связан с количеством строк первой матрицы (матрицы **A**), второй цикл (переменная j) связан с количеством столбцов второй матрицы (матрица **B**), третий цикл (переменная k) связан со смежной размерностью матриц, которая исчезает в результирующей матрице **C**. Матрицы A и B определены как неизменяемы типы (const int). Приведенный программный метод можно назвать как первый метод, метод "ijk".

Результат выполнения программы показан на рис. 5.7.

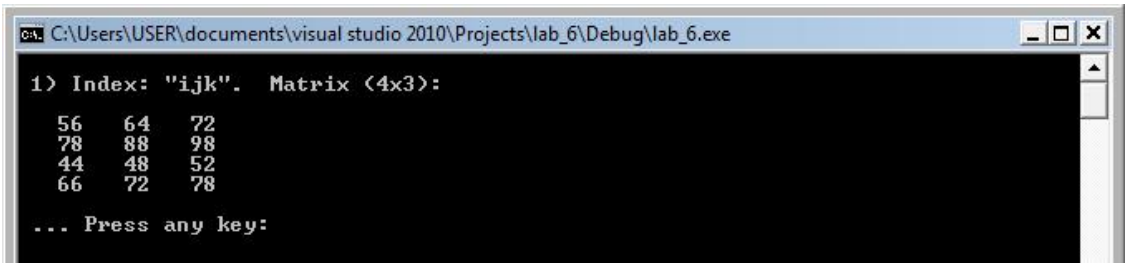


Рис. 6.4. Результат произведения двух матриц

Пример 5. *Напишите программу транспонирования матрицы, размерности которой (количество строк и количество столбцов) вводятся с клавиатуры, а элементы – вещественные случайные числа, распределенные по равномерному закону из интервала [0;15].*

По определению транспонированная матрица – это матрица A^T полученная из исходной матрицы A заменой строк на столбцы.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>
int main (void) {
      int i, j, n, m;
      double *A_ptr, *B_buf;
// Для рандомизации псевдослучайных чисел
srand((unsigned)time(NULL));
setlocale(LC_ALL, "Russian");
printf("\n Введите размерность матрицы - \n число строк
и число столбцов через пробел: ");
      scanf_s("%d%d", &n, &m);
A_ptr = (double *) calloc((n*m),sizeof(double));
B_buf = (double *) calloc((n*m),sizeof(double));
for (i = 0; i < n*m; ++i)
A_ptr[i] = 15.0*rand()/RAND_MAX;
setlocale(LC_NUMERIC, "English");
printf("\n Исходная матрица:\n");
for (i = 0; i < n; ++i) { printf("\n");
for(j = 0; j < m; ++j)
printf(" %8.4f", A_ptr[i*m+j]); }
// Основной фрагмент транспонирования
for (i = 0; i < n; ++i)
for (j = 0; j < m; ++j)
B_buf[j*n+i] = A_ptr[i*m+j];
printf("\n\n Транспонированная матрица:\n");
for (j = 0; j < m; ++j) {
      printf("\n");
for(i = 0; i < n; ++i)
printf(" %8.4f", B_buf[j*n+i]); }
// Освобождение выделенной памяти
free(A_ptr); free(B_buf);
     printf("\n\n Press any key: ");
_getch();
return 0; }
```

В программе использованы библиотечные функции для установки русских шрифтов setlocale(LC_ALL, "Russian") и вывода элементов матрицы

с плавающей точкой: setlocale(LC_NUMERIC, "English"). Для этих функций подключен заголовочный файл **locale.h**.

Возможный результат работы программы показан на рис. 6.5.

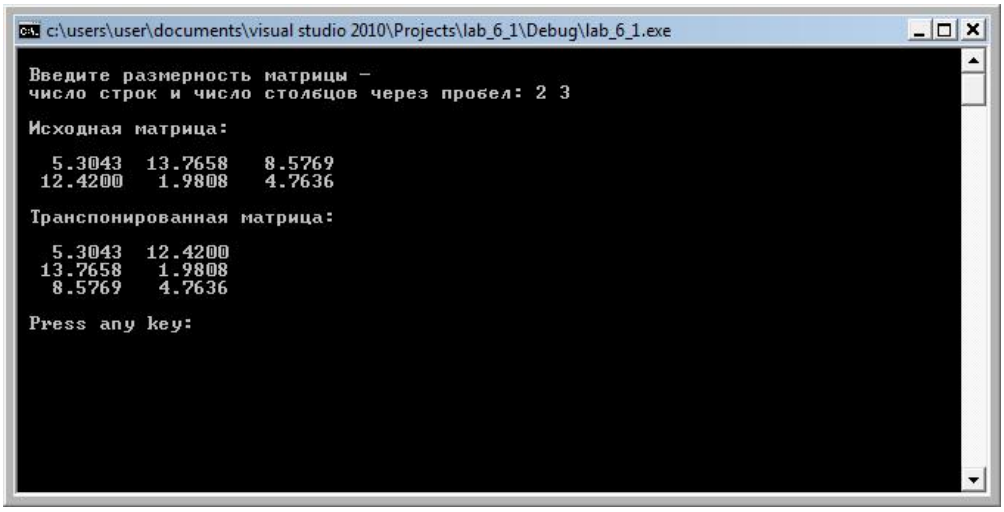


Рис. 6.5. Пример транспонирования матрицы

Пример 4. *Создать блок-схему к заданию и программу на СИ++ нахождения суммы квадратов индексов максимального элемента матрицы, ввод элементов матрицы сделать с клавиатуры.*

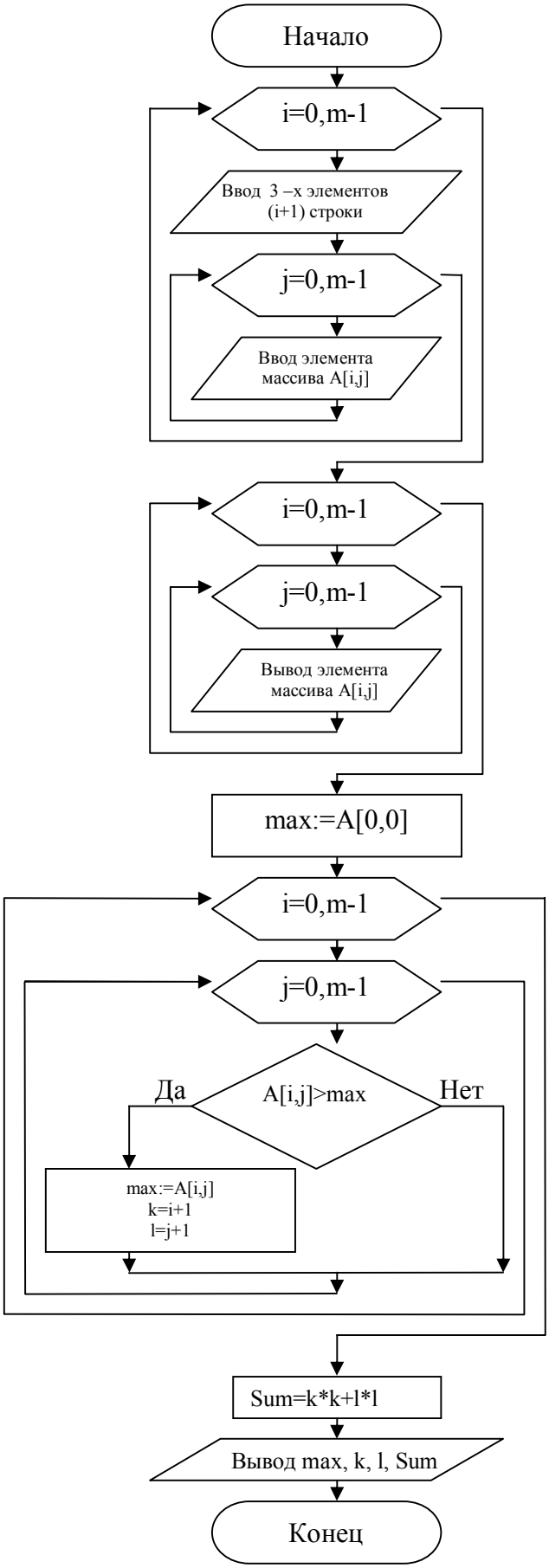


Рис.6.8. Блок-схема к заданию

Программный код решения примера:

```
//Шестая программа
//Автор Загребельный С.Л.
#include <stdio.h>
#include <conio.h>
#define m 3
int main (void) {
    int i, j, k, l,max,sum; // переменные циклов
    int A[m][m];
    // Ввод элементов матрицы
    for (i = 0; i < m; i++)
        {printf("\n vvedite 3 elementa %d
stroki\n",i+1);
        for (j = 0; j < m; j++)
            scanf("%i,%j",&A[i][j]);}
// Распечатка  матриц
    printf("\n  Ishodnaya matrica (%dx%d):\n", m, m);
    for (i = 0; i < m; i++) {
        printf("\n");
        for (j = 0; j < m; j++)
            printf(" %4d", A[i][j]);
    }
//
    max=A[0][0];
for (i = 0; i < m; i++)
        for (j = 0; j < m; j++)
            if (A[i][j]>max) {max=A[i][j];k=i+1;l=j+1;}
printf(" \n maximalniy element matrici= %d ego nomer
stroki =%d nomer stolbca=%d\n", max,k,l);
sum=k*k+l*l;
printf(" \n summa kvadratov indeksov maximalnogo
elementa matrici %d\n", sum);
    printf("\n\n ... Press any key: ");
    _getch();
    return 0;
}
```

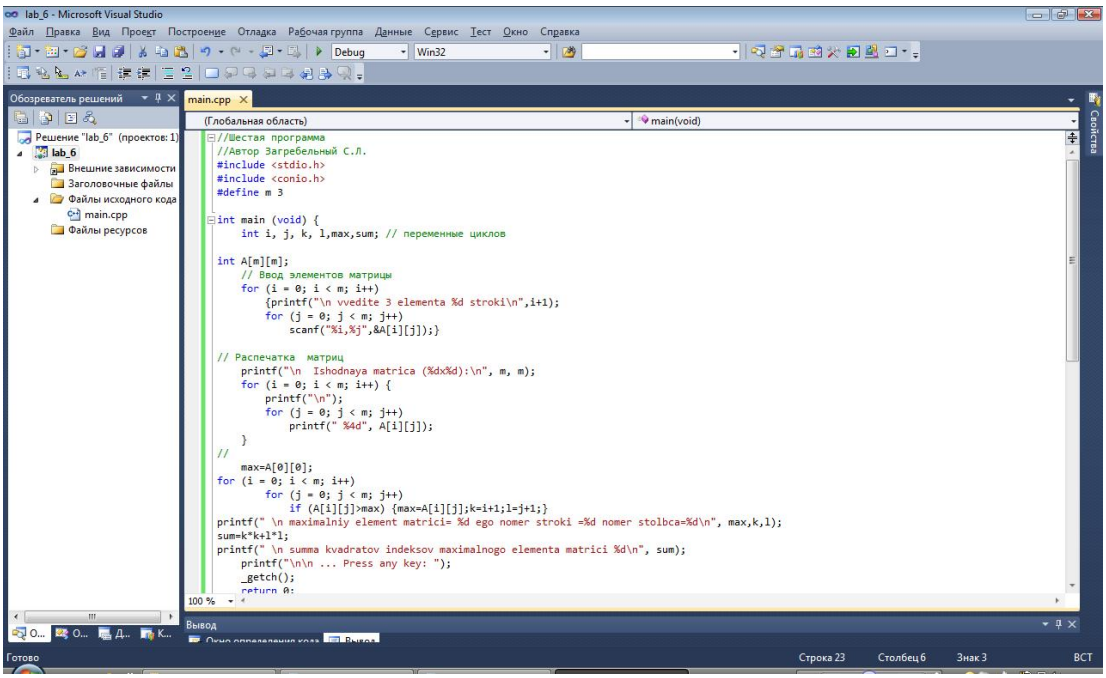


Рис.6.6. Окно примера

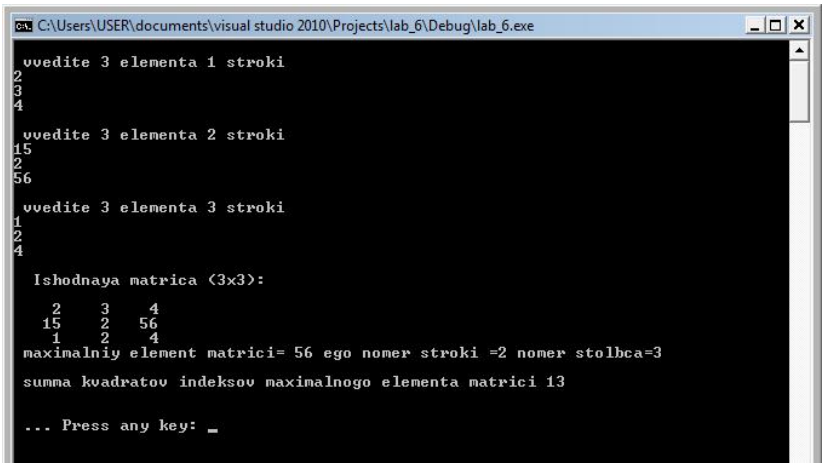


Рис.6.7. Обработка элементов матрицы

Индивидуальные задания

*Составить блок-схемы и программы для решения следующих задач таблица 6.1. (задача **а** на селективную обработку элементов матрицы, задача **б** на обработку элементов строк и столбцов)*

Таблица 6.1.

Вариант	Содержание задачи	
1	а)	Найти сумму положительных кратных 3 элементов.
	б)	Найти среднее геометрическое нечетных элементов 2-го столбца и количество кратных 5 элементов 3-ей строки матрицы D(5;5).

Продолжение таблицы 6.1.

2	а)	Найти количество отрицательных четных элементов.
	б)	Найти сумму нечетных элементов 3-го столбца и произведение отрицательных кратных 3 элементов 2-ой строки матрицы C(6;6).
3	а)	Найти произведение положительных четных элементов.
	б)	Найти произведение суммы кратных 3 чисел в 4-ом столбце на количество нечетных чисел 2-ой строки матрицы T(4;6).
4	а)	Найти сумму отрицательных четных элементов.
	б)	В матрице A(5;7) найти разность количества нечетных чисел 1-ой строки и количества четных чисел 4-го столбца.
5	а)	Найти максимальный элемент, номер строки и столбца, в котором он находится.
	б)	Найти сумму нечетных элементов 2-го столбца и произведение отрицательных кратных 3 элементов 4-ой сторки матрицы D(4;4).
6	а)	Найти минимальный элемент, номер строки и столбца, в котором он находится.
	б)	Найти произведение отрицательных четных элементов 2-ой строки и количество не кратных 5 элементов 2-го столбца матрицы B(3;5).
7	а)	Найти количество положительных кратных 5 элементов.
	б)	Подсчитать количество положительных кратных 3 элементов 1-ой строки и количество нечетных элементов 2-го столбца матрицы A(6;6).
8	а)	Найти произведение отрицательных нечетных элементов.
	б)	Найти разность произведения нечетных чисел 3-ей строки и произведения отрицательных чисел 1-го столбца матрицы B(4;4).
9	а)	Найти квадрат минимального элемента и номер строки и столбца, где он находится.
	Б)	В матрице T(3;9) найти разность произведения нечетных чисел 2-ой строки и суммы положительных чисел 6-го столбца.
10	а)	Найти произведение положительных не кратных 5 элементов.
	Б)	В матрице A(5;5) найти сумму количества четных чисел 3-ей строки и количества отрицательных чисел 4-го столбца.

Продолжение таблицы 6.1.

11	а)	Найти максимальный по модулю элемент и номер строки и столбца, где он находится.
	Б)	В матрице С(5;6) найти произведение количества нечетных чисел 2-го столбца и количества положительных чисел 3 строки.
12	а)	Найти количество отрицательных не кратных 3 элементов.
	Б)	Найти максимальный элемент 2-ой строки и количество четных элементов 5-го столбца матрицы Х(5;5).
13	а)	Найти произведение положительных нечетных элементов.
	Б)	В матрице А(6;6) найти произведение суммы четных чисел 3-ей строки и суммы отрицательных чисел 1-го столбца.
14	а)	Найти сумму отрицательных нечетных элементов.
	Б)	Найти произведение суммы положительных чисел 1-й строки на сумму четных чисел 2-го столбца матрицы М(4;5).
15	а)	Найти произведение отрицательных четных элементов.
	Б)	В матрице А(7;7) найти разность количества отрицательных чисел 2-ой строки и количества нечетных чисел 3-го столбца.
16	а)	Найти количество элементов, больших заданного числа С (ввод числа С сделать с клавиатуры).
	Б)	В матрице В(4;6) найти сумму произведения четных чисел 1-ой строки и произведения положительных чисел 3-го столбца.
17	а)	Найти минимальный по модулю элемент и номер строки и столбца, где он находится.
	Б)	Подсчитать количество кратных 3 чисел 2-ой строки и количество четных чисел 1-го столбца матрицы А(6;6).
18	а)	Найти произведение элементов, меньших заданного числа Т (ввод числа Т сделать с клавиатуры).
	Б)	Найти разность произведения нечетных чисел 3-ей строки и произведения отрицательных чисел 1-го столбца матрицы В(4;4).
19	а)	Найти сумму положительных кратных 5 элементов.
	Б)	В матрице А(8;8) найти разность произведения нечетных чисел 3-ей строки и суммы положительных чисел 6-го столбца.
20	а)	Найти произведение отрицательных четных элементов.
	Б)	В матрице А(5;5) найти сумму количества четных чисел 2-ой строки и количества отрицательных чисел 4-го столбца.

Продолжение таблицы 6.1.

21	а)	Найти количество положительных нечетных элементов.
	Б)	В матрице A(3;3) найти произведение количества нечетных чисел 1-ой строки и количества положительных чисел 3-го столбца.
22	а)	Найти количество элементов, меньших числа 5.
	Б)	Найти максимальный элемент 3-го столбца и сумму нечетных элементов 1-ой строки матрицы T(5;5).
23	а)	Найти произведение положительных кратных 3 элементов.
	Б)	В матрице A(6;6) найти произведение суммы четных чисел в 3-ей строке и суммы отрицательных чисел 1-го столбца.
24	а)	Найти сумму отрицательных не кратных 5 элементов.
	Б)	Найти произведение суммы положительных чисел в 4-ом столбце на количество четных чисел 2-ой строки матрицы F(6;6).
25	а)	Найти квадрат максимального элемента и номер строки и столбца, где он находится.
	Б)	В матрице A(7;7) найти разность количества положительных чисел 1-ой строки и количества четных чисел 3-го столбца.
26	а)	Найти сумму четных элементов из интервала [-10;10] матрицы A(4;4).
	Б)	Найти произведение количества четных элементов 3 строки на сумму нечетных элементов 2 столбца матрицы B(5;4).
27	а)	Найти количество кратных 3 элементов из интервала [-6;8] матрицы A(5;5).
	б)	В матрице A(3;3) найти произведение количества четных чисел 2-ой строки и количества отрицательных чисел 4-го столбца.
28	а)	Найти произведение отрицательных нечетных элементов матрицы A(4;3).
	б)	В матрице A(5;5) найти произведение количества нечетных чисел 3-го столбца и количества отрицательных чисел 3 строки.

Продолжение таблицы 6.1.

29	а)	Найти количество положительных элементов из интервала [-5;6] матрицы B(6;6).
	б)	В матрице A(6;6) найти произведение суммы кратных 3 чисел 2-ей строки и суммы отрицательных чисел 2-го столбца.
30	а)	Найти максимальный по модулю элемент и номер строки и столбца, где он находится.
	б)	Найти произведение количества четных чисел в 2-ом столбце на количество нечетных чисел 2-ой строки матрицы B(4;4).

Контрольные вопросы

1. Как организуются многомерные числовые массивы в языке **C**?
2. Как организуется индексирование числовых массивов в языке **C**?
3. На кого или на что возлагается контроль границ числовых массивов в языке программирования **C**?
4. В какой очередности и как происходит заполнение многомерных числовых массивов в программах на языке **C**?
5. Для чего применяется начальная инициализация числовых массивов при дальнейшем их использовании?
6. Сколько потребуется операторов цикла для вывода на консоль двухмерного числового массива (матрицы чисел)?
7. Почему при определении размерности массива с помощью препроцессорной директивы **define** не используется точка с запятой после числового значения?

ЛАБОРАТОРНАЯ РАБОТА 7

Построение графика функции

Теоретическая часть

Принципы программирования на языке **C** основаны на понятии функции. Например, к системным функциям относятся printf(), scanf(), gets(), putchar() и др. Функции – это строительные элементы языка **C** и то место, в котором выполняется вся работа программы.

Большие программы обычно состоят из нескольких пользовательских функций и ряда системных функций. Функция – самостоятельная единица программы. Функции повышают уровень модульности программы, облегчают ее чтение, внесение изменений и коррекцию ошибок.

В основе всех программ на языке программирования **C** лежат одни и те же фундаментальные элементы – функции. В частности, функция main() является обязательной для любой программы. Во всех программах **C** определяется единая внешняя функция с именем main(), служащая точкой входа в программу, то есть первой функцией, выполняемой после запуска программы.

Ни одна программа в языке **C++** не может обойтись без функций.

Функция в языке **C** играет ту же роль, что и подпрограммы или процедуры в других языках. Каждая функция языка **C** имеет имя и список аргументов. По соглашению, принятому в языке **C**, при записи имени функции после него ставятся круглые скобки. Это соглашение позволяет легко отличить имена переменных от имен функций.

Рассмотрим модельный пример программы, в которой, кроме функции main(), содержатся еще три функции.

```
#include <stdio.h>
int main(void)  /* Главная функция */
{  /* Начало тела функции */
function1(); /* вызов первой функции */
function2(); /* вызов второй функции */
function3(); /* вызов третьей функции */
} /* Конец тела функции main() */

/* Начало определения первой функции */
function1() { /* Начало тела первой функции */
/* Операторы первой функции */
/* Конец тела первой функции */
}

/* Начало определения второй функции */
function2()
{ /* Начало тела второй функции*/
```

```
/* Операторы второй функции */
/* Конец тела второй функции*/
}

/* Начало определения третьей функции */
function3()
{ /* Начало тела третьей функции*/
/* Операторы третьей функции */
/* Конец тела третьей функции*/
}
```

В условной (модельной) программе имеются четыре функции: main(), function1(), function2(), function3(). Эти функции не имеют аргументов. Позднее рассмотрим функции, которые имеют аргументы. Аргументы функции – это величины, которые передаются функции во время ее вызова. Аргумент, стоящий в операторе вызова функции, называется фактическим параметром. Аргументы, стоящие в заголовке функции, называются формальными параметрами. В языке **C++** функция может возвращать значение в вызывающую программу посредством оператора return. Оператор возврата из функции в точку вызова имеет две формы:

```
return;
return выражение;
```

В общем виде функция выглядит следующим образом:

```
возвр-тип  имя-функции(список параметров)
{
Тело_функции
}
```

Тело_функции – это часть определения функции, ограниченная фигурными скобками и непосредственно размещенная вслед за заголовком функции. Тело функции может быть либо составным оператором, либо блоком. В языке С определения функций не могут быть вложенными, т.е. внутри одной функции нельзя объявить и расписать тело другой функции.

Возвращаемый тип **возвр-тип** функции определяет тип данного, возвращаемого функцией. Например, это могут быть int, float, double и т.д. В случае, когда функция ничего не возвращает, ей присваивается тип void.

Функция может возвращать любой тип данных, за исключением массивов список параметров – это список, элементы которого отделяются друг от друга запятыми. При вызове функции параметры принимают значения аргументов. Если функция без параметров, то такой пустой список можно указать в явном виде, поместив для этого внутри скобок ключевое слово void. Все параметры функции (входящие в список параметров) должны объявляться отдельно, причем для каждого из них надо указывать и тип, и имя. В общем виде список объявлений параметров должен выглядеть следующим образом:

```
fun(тип имя_перем1, тип имя_перем2,..., тип имя_перем N)
```

Например:

```
fun(int i, int j, float k, char str1, char str2)
```

Рассмотрим пример программы с выводом сообщения не в главной функции main(), а в другой:

```
#include <stdio.h>
#include <conio.h>

void printMessage (void)
{
printf("\n\t hello, world\n");
return;
printf("\n\t 123\n");
}

int main(void)
{
printMessage();

printf("\n Press any key: ");
_getch();
return 0;
}
```

Результат выполнения программы показан на рис. 7.1.

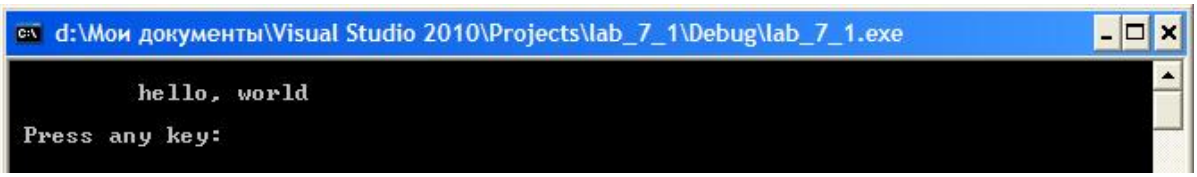


Рис. 7.1. Вывод сообщения с помощью двух функций

Программа состоит из двух функций: printMessage() и main(). Выполнение программы всегда начинается с функции main(), которую называют еще главной. Внутри функции main() происходит вызов функции printMessage() без параметров. Когда происходит вызов функции, выполнение программы передается непосредственно вызванной функции. Внутри функции printMessage() выполняется только утверждение

```
printf("\n\t hello, world\n");
```

Несмотря на то, что в функции printMessage() есть еще одно утверждение printf("\n\t 123\n"), которое не выполняется, поскольку используется утверждение возврата (return) из функции.

В языке **C** функция введена как один из производных типов.

Формальные параметры в определениях функций могут объявляться в

форме прототипа. Прототипы дают компилятору возможность тщательнее выполнять проверку типов аргументов. Если используются прототипы, то компилятор может обнаружить любые сомнительные преобразования типов аргументов, необходимые при вызове функции, если тип ее параметров отличается от типов аргументов. Компилятор также обнаружит различия в количестве аргументов, использованных при вызове функции, и в количестве параметров функции.

В общем случае прототип функции должен выглядеть таким образом:

```
тип имя_функции(тип имя_парам1, тип имя_парам2,..., тип им_парамN);
```

В приведенной выше программе прототип функции printMessage() не использовался, так как сама функция была объявлена до главной функции main(). Для переносимости **C**-кода в **C**++ использование прототипа функции обязательно. Поэтому к хорошему стилю программирования относится использование прототипов функций, поскольку большие программы обычно состоят из нескольких функциях, часто расположенных в различных файлах.

Вышеприведенная программа с использованием прототипа функции printMessage() будет выглядеть следующим образом:

```
#include <stdio.h>
#include <conio.h>

//void printMessage (void);//Прототип функции

int main(void) {
void printMessage (void); //Прототип функции
printMessage(); // Вызов функции
printf("\n Press any key: ");
_getch();
return 0;
 }

// Определение функции
void printMessage (void)
{
printf("\n\t hello, world\n");
return;
printf("\n\t 123\n");
 }
```

В листинге программы показаны две возможности использования прототипа функции printMessage(). При этом, сама функция printMessage() объявлена после функции main().

Формальные параметры функции определены в прототипе функции. При обращении к функции используются фактические параметры, называемые аргументами функции.

Список фактических параметров – это список выражений, количество которых равно количеству формальных параметров функции (исключение составляют функции с переменным числом параметров). Соответствие между формальными и фактическими параметрами устанавливается по их взаимному расположению в списках. Между формальными и фактическими параметрами должно быть соответствие по типам.

Синтаксис языка **C++** предусматривает только один способ передачи параметров – передачу по значениям. Это означает, что формальные параметры функции локализованы в ней, т.е. недоступны вне определения функции и никакие операции над формальными параметрами в теле функции не изменяют значений фактических параметров.

Передача параметров по значению предусматривает следующие шаги:

1. При компиляции функции выделяются участки памяти для формальных параметров, т.е. формальные параметры оказываются внутренними объектами функции. При этом для параметров типа float формируются объекты типа double, а для параметров типов char и short int создаются объекты типа **int**. Если параметром является массив, то формируется указатель на начало этого массива, и он служит представлением массива-параметра в теле функции.
2. Вычисляются значения выражений, использованных в качесстве фактических параметров при вызове функции.
3. Значения выражений – фактических параметров заносятся в участки памяти, выделенные для формальных параметров функции.
4. В теле функции выполняется обработка с использованием значений внутренних объектов-параметров, и результат передается в точку вызова функции как возвращаемое ею значение.
5. Никакого влияния на фактические параметры (на их значения) функция не оказывает.
6. После выхода из функции освобождается память, выделенная для ее формальных параметров.

Важным является момент, что объект вызывающей программы, использованный в качестве фактического параметра, не может быть изменен из тела функции. Для подобного изменения существует косвенная возможность изменять значения объектов вызывающей программы действиями в вызванной функции. Это становится возможным с помощью указателя (указателей), когда в вызываемую функцию передается адрес любого объекта из вызывающей программы. С помощью выполняемого в тексте функции разыменования указателя осуществляется доступ к адресуемому указателем объекту из вызывающей программы. Тем самым, не изменяя самого параметра (указатель-параметр постоянно содержит только адрес одного и того объекта), можно изменять объект вызывающей программы.

Массивы и строки также могут быть параметрами функции. В этом случае внутрь функции передается только адрес начала массива. Тогда можно в качесстве параметра использовать указатель. Приведем два

равноправных прототипа функций:

```
float fun(int n, float A[ ], float B[ ]);
float fun(int n, float *a, float *b);
```

Поскольку массив передается в функцию как указатель, внутри функции можно изменять значения элементов массива–фактического параметра, определенного в вызывающей программе. Это возможно и при использовании индексирования, и при разыменовании указателей на элементы массива.

В языке С существует возможность создавать функции, число аргументов которых не определено – функции с переменным числом аргументов. При этом следует указать только количество аргументов. Пример прототипа функции с переменным числом аргументов:

```
int fun(int n, j);
```

Многоточие (j) в прототипе функции означает, что функция получает переменное число аргументов любого типа. Многоточие должно всегда находиться в конце списка параметров.

Макросы и определения заголовочного файла переменных аргументов stdarg.h (табл. 7.1) предоставляют программисту средства, необходимые для построения функций со списком аргументов переменной длины.

Таблица 7.1. Макросы заголовочного файла stdarg.h

Идентификатор	**Объяснение**
va_list	Тип, предназначающийся для хранения информации, необходимой макросам v_start, va_arg и va_end. Чтобы получить доступ к аргументам в списке переменной длины, необходимо объявить объект типа va_list
va_start	Макрос, который вызывается перед обращением к аргументам списка переменной длины. Он инициализирует объект, объявленный с помощью va_list, для использования макросами va_arg и va_end
va_arg	Макрос, расширяющийся до выражения со значением и типом следующего аргументов списке переменной длины. Каждый вызов его изменяет объект, объявленный с помощью va_list так, что объект указаывает на следующий аргумент списка
va_end	Макрос обеспечивает нормальный возврат из функции, на список аргументов которой ссылается макрос va_start

Примеры обращений к функции с фактическими аргументами:

```
double k;
```

```
    double v1 = 1.5,
           v2 = 2.5,
           v3 = 3.5;
// Первый вариант, где 3 – количество аргументов
           k = fun(3,v1, v2, v3);
// Второй вариант, где 0.0 – завершающий нуль списка
аргументов
      k = fun(v1, v2, v3, 0.0);
```

Указатели, передаваемые в функцию, могут быть указателями на указатели. Указатели могут указывать на начало какого-либо массива и т. д. Указатели могут использоваться для защиты массивов, над которыми необходимо произвести некоторые вычисления или преобразования.

Особым свойством указателей можно считать возможность использовать их в качестве возвращаемых значений функций. Поскольку функции возвращают только одно значение, то несколько значений одного типа можно поместить в массив, а затем указатель на этот массив использовать в качестве возвращаемого значения.

Общая форма определения функции, которая возвращает указатель, следующая:

```
тип  *имя_функции ( аргументы функции )
{
// тело функции

тип *имя_указателя;
?

return  имя_указателя;
}
```

Рассмотрим пример, в котором осуществляется сложение двух одномерных массивов и результат возвращается через указатель.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int *out2(int A[], int B[], int);

int main (void) {
     int i, n;
     int A[] = {1,2,3,4,5};
     int B[] = {2,2,2,2,2};
     int *ptrAB = NULL;
```

```
      n = (sizeof(A)/sizeof(A[0]));

      puts("\n The initial arrays: ");
      for (i = 0; i < n; i++)
            printf(" %d", A[i]);

     puts("");
      for (i = 0; i < n; i++)
            printf(" %d", B[i]);

      ptrAB = out2(A, B, n);
      puts("\n\n Result from function: ");
      for (i = 0; i < n; i++)
            printf(" %d", ptrAB[i]);
      puts("\n\n Control of the arrays: ");
      for (i = 0; i < n; i++)
            printf(" %d", A[i]);

     puts("");
      for (i = 0; i < n; i++)
            printf(" %d", B[i]);
     free(ptrAB); // освобождение выделенной памяти

      printf("\n\n ... Press any key: ");
      _getch();
      return 0;
}

int *out2(int A[], int B[], int n)
{
     int i;
int *ptr = (int *)calloc(n, sizeof(int)); //выделение
памяти

     for (i = 0; i < n; i++)
     ptr[i] = A[i] + B[i];

     return ptr;
}
```

Программа не требует особых пояснений.

Следует отметить, что никогда не следует возвращать адрес переменной, определенной в теле функции, так как переменные функции являются локальными, и они существуют только во время работы функции.

Указатели возвращаются подобно значениям любых других типов данных. Чтобы вернуть указатель, функция должна объявить его тип в

качестве типа возвращаемого значения. Таким образом, если функция возвращает указатель, то значение, используемое в ее инструкции return, также должно быть указателем. В частности, многие библиотечные функции, предназначенные для обработки строк, возвращают указатели на символы.

В языке **C++** существует такой механизм как указатель на функцию. Допустим, существует несколько функций для различных операций с данными. В этом случае оказывается удобным определить указатель на функцию, и использовать его там, где требуется производить расчет для различных функций.

Указатель на функцию – это переменная, содержащая адрес в памяти, по которому расположена функция. Имя функции – это адрес начала программного кода функции. Указатели на функции могут быть переданы функциям в качестве аргументов, могут возвращаться функциями, сохраняться в массивах и присваиваться другим указателям на функции.

Типичное определение указателя на функцию следующее:

```
тип_возвращаемый_функцией(*имя_указателя_на_функцию)(аргументы);
```

В приведенном объявлении используются круглые скобки, в которых собственно и определяется указатель на функцию, которая возвращает тот или иной тип – тип_возвращаемый_функцией. Хотя знак * обозначает префиксную операцию, он имеет более низкий приоритет, чем функциональные круглые функции, поэтому для правильного комбинирования частей объявления необходимы еще и дополнительные скобки. При этом аргументы – это аргументы той или иной функции с заданным типом возвращаемого значения, и на которую ссылается указатель *имя_указателя_на_функцию. Очевидно, что возможны сложные объявлений функций.

Указатели на функции часто используются в системах, управляемых меню. Пользователь выбирает команду меню (одну из нескольких). Каждая команда обслуживается своей функцией. Указатели на каждую функцию находятся в массиве указателей. Выбор пользователя служит индексом, по которому из массива выбирается указатель на нужную функцию.

Другим типичным применением указателей на функции являются реализация обобщенных алгоритмов, например, алгоритмов сортировки и поиска. В этом случае критерии сортировки и поиска реализуются в виде отдельных функций и передаются при помощи указателей на функции в качестве параметра реализации основного алгоритма.

Практическая часть

Пример 1. *Напишите программу сортировки по возрастанию заданного массива случайных чисел, равномерно распределенных в интервале [–6;6], с помощью вспомогательной функции.*

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 10

// Прототип функции с формальными параметрами
void sort(double arr[], int n);

int main (void) {
 double M[MAX];
 int i, size = MAX;
 long int L;
 unsigned int some;
 L = (long) time(NULL);
 srand((unsigned)L);

 for (i = 0; i < MAX; ++i)
      M[i] = 12.0*rand()/RAND_MAX - 6.0;

 printf("\n\t The original array:\n");
 for (i = 0; i < MAX; ++i)
      printf("\t%8.4f\n", M[i]);

// Обращение к функции с фактическими параметрами
 sort(M, size);
// Распечатка отсортированного массива
 printf("\n\t After sorting: \n");
for (i = 0; i < MAX; ++i) printf("\t%8.4f\n",
M[i]);
 printf("\n Press any key: ");
     _getch();
    return 0; }
// Вспомогательная функция сортировки
void sort(double Array[], int m) {
int i, j;
double tmp;
for (i = 0; i < m-1; ++i)
   for (j = 0; j < m-i-1; ++j)
if (Array[j+1] < Array[j]) {
tmp = Array[j];
Array[j] = Array[j+1];
```

```
Array[j+1] = tmp;
}
}
```

Следует обратить внимание на имена формальных параметров в самой функции sort() и в ее прототипе: они имеют разные имена, но одинаковые типы. Фактические параметры или аргументы функции sort() в вызывающей программе (в теле функции main()) имеют свои имена, не связанные с именами формальных параметров.

Заполнение массива случайными числами производится с помощью библиотечной функции rand() и макроопределения RAND_MAX. Для рандомизации массива случайных чисел при каждом новом запуске программы используется библиотечная функция srand(), аргументом которой является системное время, формируемое библиотечной функцией time().

Возможный результат выполнения программы показан на рис. 7.2.

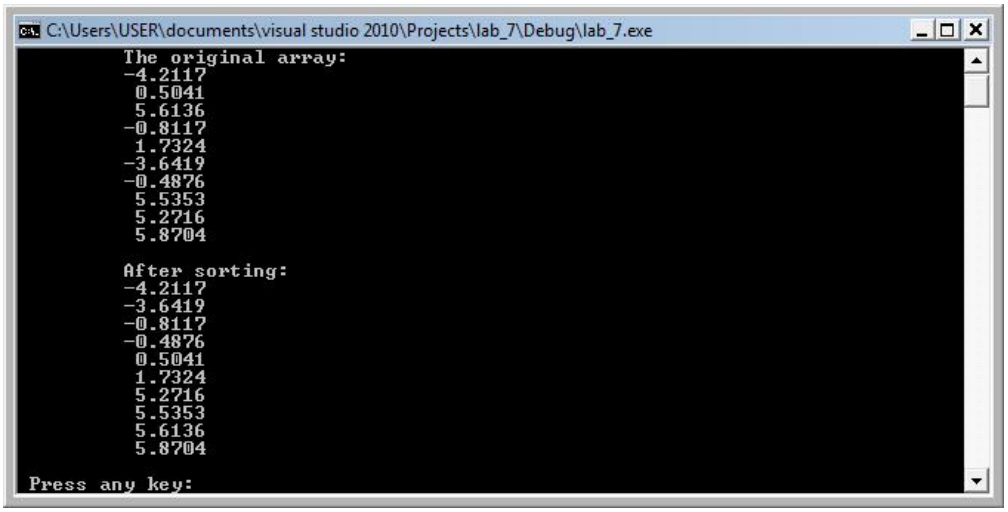


Рис. 7.2. Пример сортировки числового массива

Ранее было отмечено, что в языке **C++** аргументы передаются в функции по значению и не существует прямого способа изменить переменную вызывающей функции, действуя внутри вызываемой функции. Благодаря аргументам-указателям функция может обращаться к объектам в вызвавшей ее функции, в том числе модифицировать их. В качестве примера рассмотрим функцию swap(), в задачу которой входит обмен элементов местами. Для решения такой задачи необходимо передать из вызывающей программы (например, из главной функции main()) в функцию указатели на переменные, которые нужно изменить. Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

// Прототип функции
```

```
void swap(int*, int*);

int main (void) {
      int a = 10,
          b = -20;

    // Вывод на консоль исходных значений переменных
    printf("\n Initial values:\n a = %d, b = %d\n", a,
b);

    // Вызов функции swap() с фактическими параметрами
     swap(&a, &b);

    // Результат после обращения функции swap()
     printf("\n New values:\n a = %d, b = %d\n", a, b);

      printf("\n ... Press any key: ");
      _getch();
      return 0;
}

// Определение функции
void swap(int *pa, int *pb)
{
int temp;
temp = *pa;
*pa = *pb;
*pb = temp;
}
```

В программе в качестве фактических параметров функции swap() выступают адреса заданных переменных. Можно было в главной функции определить указатели и инициализировать их адресами заданных переменных, а потом передать эти указатели в функцию swap.

Результат выполнения программы показан на рис. 7.3.

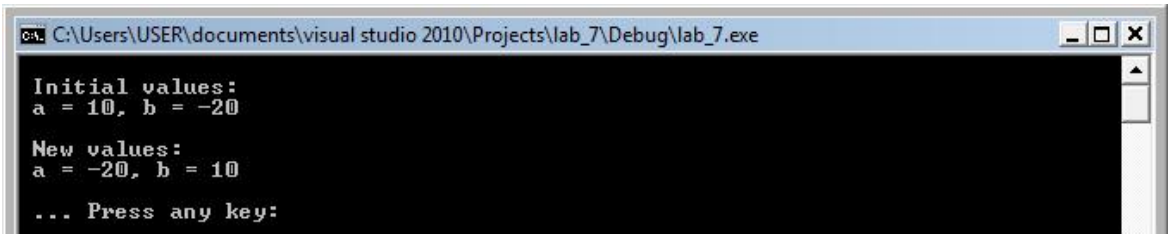


Рис. 7.3. Результат обмена данными, выполненного функцией swap()

Пример 4. *Напишите программу с функцией пузырьковой сортировки,*

использующей вызов по ссылке.

В условии примера "вызов по ссылке" означает, что в качестве фактических параметров функций будут использоваться адреса переменных. И в этом случае прототип таких функций будет содержать указатели на соответствующие типы.

Программный код решения примера:

```
#include <stdio.h>
#include <conio.h>

// Прототип функции
void bsort (int* const, const int);

int main (void)
{
      int A[] = {56, 34, 2, 0, 1, -21, 6, 8, 7};
      int i, n;
          //Размерность массива
      n = sizeof(A)/sizeof(A[0]);
      puts("\n Data items in original order:");
      for (i = 0; i < n; i++)
           printf(" %3d", A[i]);

      // Вызов функции сортировки - bsort()
      bsort (A, n);
       puts("\n\n Data items in ascending order:");
      for (i = 0; i < n; i++)
       printf(" %3d", A[i]);
            printf("\n\n ... Press any key: ");
      _getch();
      return 0;
}
// Определение функции
void swap(int *pa, int *pb)
{
int temp;
temp = *pa;
*pa = *pb;
*pb = temp;
}


void bsort (int *const arr, const int size)
```

```
{
int pass, //счетчик проходов
       j; // счетчик сравнений
// Прототип функции обмена - swap()
void swap (int*, int*);
// Цикл для контроля проходов
for (pass = 0; pass < size - 1; pass++ )
{
      // цикл для контроля сравнений на данном проходе
      for (j = 0; j < size - 1; j++)
      {
     //   обмен   значений   при   нарушении   порядка
возрастания
            if (arr[j] > arr[j + 1])
            {
            swap(&arr[j], &arr[j+1]);
            }
      }
}
}
```

В программе функция сортировки bsort() в качестве формального параметра используется константный указатель, который указывает на первый элемент заданного массива. Второй формальный параметр также константный, чтобы подчеркнуть неизменность этого параметра в теле функции bsort(). Передача функции размера массива в качестве параметра имеет два преимущества – это хороший стиль программирования и, кроме того, такую функцию можно использовать многократно.

Прототип функции swap() включен в тело функции bsort(), потому что это единственная функция, которая вызывает функцию обмена swap().

Пример выполнения программы показан на рис. 7.4.

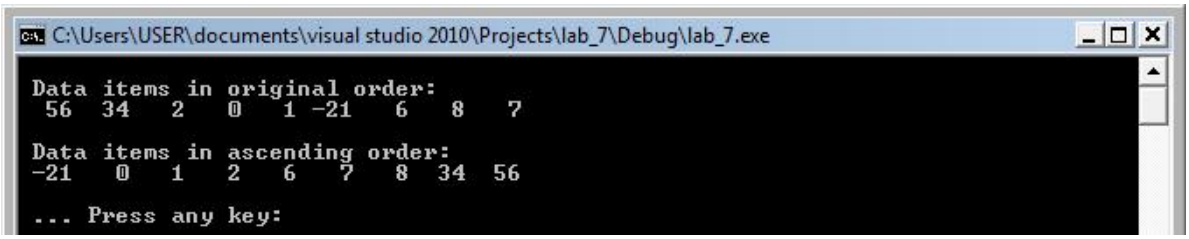


Рис. 7.4. Пример сортировки массива методом пузырька

Пример 5. *Напишите программу построения на экране дисплея графика следующей функции:*

$$y = \sin(3x)e^{x/3}$$

Предусмотрите возможность записи в текстовый файл графика данной

функции.

Для решения примера используем средства вывода на печать форматированных данных без применения специальных графических библиотек.

Программный код решения примера:

```
// Заголовочные файлы
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
// Размеры диаграммы по ширине и высоте экрана
#define SCREENW  79
#define SCREENH  25
// Функция построения графика заданной функции
void plot (FILE *fout, double a, double b, double (*f)
(double))
{
// Формальные параметры функции plot
// FILE *fout – указатель на поток вывода
// double a – левая граница оси абсцисс
// double b – правая граница оси абсцисс
// double (*f) (double) – указатель на функцию
     char screen[SCREENW][SCREENH];
     double x, y[SCREENW];
     double ymin = 0, ymax = 0;
     double hx, hy;
     int i, j;
     int xz, yz;
// hx – шаг по оси абсцисс
     hx = (b - a) / (SCREENW - 1);
     for (i = 0, x = a; i < SCREENW; ++i, x += hx) {
// вычисляем значение функции
     y[i] = f (x);
// запоминаем минимальное и максимальное значения
     if (y[i] < ymin) ymin = y[i];
     if (y[i] > ymax) ymax = y[i];
     }
     hy = (ymax - ymin) / (SCREENH - 1);
     yz = (int)floor (ymax / hy + 0.5);
     xz = (int)floor (-a / hx + 0.5);

// рисование осей координат
     for (j = 0; j < SCREENH; ++j) {
```

```
            for (i = 0; i < SCREENW; ++i) {
                  if (j == yz && i == xz)
                        screen[i][j] = '+';
                  else if (j == yz)
                        screen[i][j] = '-';
                  else if (i == xz)
                        screen[i][j] = '|';
                  else
                        screen[i][j] = ' ';
            }
      }
// рисование графика функции
      for (i = 0; i < SCREENW; ++i) {
            j = (int)floor ((ymax - y[i]) / hy + 0.5);
     screen[i][j] = '.'; // символ начертания графика
      }
//  вывод результата в файл или в стандартный поток
stdout
      for (j = 0; j < SCREENH; ++j) {
            for (i = 0; i < SCREENW; ++i)
                  fputc (screen[i][j], fout);
            fprintf (fout, "\n");
      }
}
// Заданная функция
double f (double x)
{
      return sin (3.0*x) * exp (-x / 3.0);
}
int main (void)
{
// Вывод графика в стандартный поток (консоль)
      plot (stdout, 0.0, 10.0, f);
      printf("\n\n … Press any key: ");
      _getch();
      return 0;
}
```

В программе используется указатель на файл, который может быть стандартным потоком, т. е. экран дисплея. В главной функции main() происходит обращение к функции рисования графика plot(), в которую вводят фактические параметры, в частности файл – это stdout, т. е. стандартный поток, 0.0 – это левая граница оси абсцисс, 10.0 – правая граница оси абсцисс, f – имя функции с описанием зависимости y = f(x).

Пример выполнения программы показан на рис. 7.5.

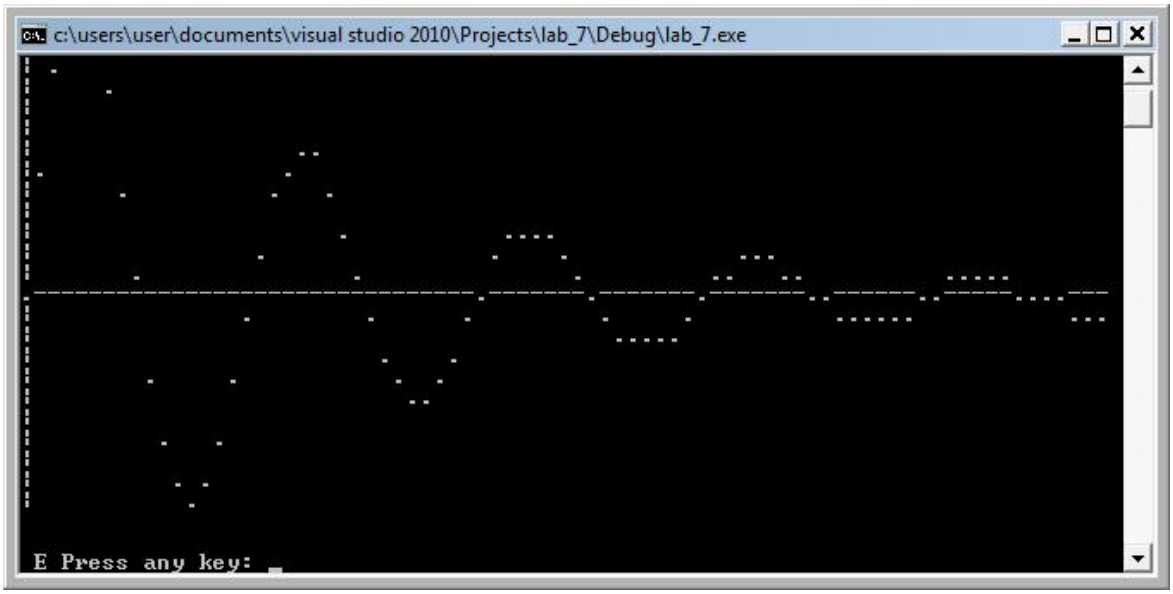


Рис. 7.5. Пример построения графика функции на консоли

Индивидуальные задания

Согласно своему варианту (таблица 7.2.) создать программу для построения графика функции.

Таблица 7.2.

Вариант	Функция, для построения графика
1	$\sin^3 2x$
2	$2\sqrt{x^3}\sin x^3$
3	$\sqrt[3]{x^3}\sin x$
4	$5x^2\sin^3 x^3 + 2x^3$
5	$x^3\cos^2\left(x^5 + 2x\right)$
6	$3x^3\sin^2 x^5$
7	$\ln(\sin 4x + 1)^2$
8	$\sin^3 x^2$
9	$\cos^3\left(x^3 + 2\right)^2$
10	$x^2\cos x$
11	$\sin\left(\sqrt{x^5 + 2x}\right)$
12	$(x-1)^3 + \cos 2x^3$

Продолжение таблицы 7.2.

13	$\cos\left(x^5+2x^3-x\right)^5$
14	$x\sin\left(x^2+2x\right)$
15	$\ln(\cos 2x+1)^3$
16	$5^{x+1}\sin\left(x^3+1\right)$
17	$3\ln\sqrt[5]{\sin x+x^2}$
18	$\sin x^{2x}+\cos\left(x^2+2\right)$
19	$x^x-\cos x$
20	$x^2+\sin 5x$
21	$\sqrt{x^3-1}+\sin x^2$
22	$2\sin(x-e^{-x})$
23	$\sin x^2+x^{0,25}$
24	$\sqrt[3]{\sin^2 x+\cos^4 x}$
25	$\sin^3 x^4$
26	$\sin\sqrt[3]{x^3+x^2}$
27	$x^x\cos x$
28	$x^4\sin 4x$
29	$\sin^2 x^3$
30	$(x+1)^2\cos x^3$

Контрольные вопросы

1. Каким образом можно вернуть из функции несколько значений?
2. Каким образом определяется тип функции?
3. Как выглядит описание функции, которая возвращает указатель на заданный тип, например, char?
4. В каком месте программы можно определить указатель на функцию?
5. Имеет ли указатель на функцию прототип и определение?
6. Как осуществляется вызов функции с помощью указателя?
7. Как взаимосвязаны между собой объявление функции, ее определение и вызов функции?

ЛАБОРАТОРНАЯ РАБОТА 8

Файловый ввод и вывод в языке C++

Теоретическая часть

Файл – это именованный объект, хранящий данные (программа или любая другая информация) на каком-либо носителе (дискета, винчестер, CD).

В языке **C++** файлом может быть все что угодно, начиная с дискового файла и заканчивая терминалом или принтером. Поток связывают с определенным файлом, выполняя операцию открытия. Как только файл открыт, можно проводить обмен информацией между ним и программой.

Не у всех файлов одинаковые возможности. Например, к дисковому файлу прямой доступ возможен, в то время как к некоторым принтерам – нет. В языке **C++** все потоки одинаковы, а файлы – нет.

Если файл может поддерживать запросы на местоположение (указатель текущей позиции), то при открытии такого файла указатель текущей позиции в файле устанавливается в начало. При чтении из файла (или записи в него) каждого символа указатель текущей позиции увеличивается, обеспечивая тем самым продвижение по файлу.

Файл отсоединяется от определенного потока (т.е. разрывается связь между файлом и потоком) с помощью операции закрытия. При закрытии файла, открытого с целью вывода, содержимое (если оно есть) связанного с ним потока записывается на внешнее устройство. Этот процесс, который обычно называют дозаписью потока, гарантирует, что никакая информация случайно не останется в буфере диска. Если программа завершает работу нормально, т.е. либо функция main() возвращает управление операционной системе, либо вызывается функция exit(), то все файлы закрываются автоматически. В случае аварийного завершения программы, например, в случае краха или завершения путем вызова функции abort(), файлы не закрываются.

Файловая системы языка **C++** предназначена для работы с самыми разнообразными устройствами, в том числе терминалами, дисками и накопителями на магнитной ленте. Даже если какое-то устройство сильно отличается от других, буферизованная файловая система все равно представит его в виде логического устройства, которое называется потоком. Потоки бывают двух видов: текстовые и двоичные.

Текстовый поток – это последовательность символов. В стандарте **C++** считается, что текстовый поток организован в виде строк, каждая из которых заканчивается символом новой строки. Однако в конце последней строки этот символ не является обязательным. В текстовом потоке по требованию базовой среды могут происходить определенные преобразования символов. Например, символ новой строки может быть заменен парой символов – возврата каретки (например, \r) и перевода строки (например, \n), т.е. \r\n.

Двоичные потоки – это последовательность байтов, которая взаимно однозначно соответствует байтам на внешнем устройстве, причем никакого

преобразования символов не происходит. Кроме того, количество тех байтов, которые пишутся (читаются), и тех, которые хранятся на внешнем устройстве, одинаково. Однако в конце двоичного потока может добавляться определяемое приложением количество нулевых байтов. Такие нулевые байты, например, могут использоваться для заполнения свободного места в блоке памяти незначащей информацией, чтобы она в точности заполнила сектор на диске.

Файловая система языка **C++** состоит из нескольких взаимосвязанных функций. Самые распространенные из них показаны в табл. 8.1.

Таблица 8.1.

Функции файловой системы языка C

№ п/п	Имя функции	Что делает
1.	fopen()	Открывает файл
2.	fclose()	Закрывает файл
3.	putc()	Записывает символ в файл
4.	fputc()	То же, что и putc()
5.	getc()	Читает символ из файла
6.	fgetc()	То же, что и getc()
7.	fgets()	Читает строку из файла
8.	fputs()	Записывает строку в файл
9.	fseek()	Устанавливает указатель текущей позиции на определенный байт файла
10.	ftell()	Возвращает текущее значение указателя текущей позиции в файле
11.	fprintf()	Для файла то же, что printf() для консоли
12.	fscanf()	Для файла то же, что scanf() для консоли
13.	feof()	Возвращает значение true (истина), если достигнут конец файла
14.	ferror()	Возвращает значение true (истина), если произошла ошибка
15.	rewind()	Устанавливает указатель текущей позиции в начало файла
16.	remove()	Стирает файл
17.	fflush()	Дозапись потока в файл

Для приведенных функций требуется подключить заголовок <stdio.h>. Запись или чтение из файла осуществляются с помощью указателя файла. Указатель файла – это указатель на структуру типа FILE. Для объявления переменной–указателя файла, например, *fp, используется следующий оператор:

```
FILE  *fp;
```

Ключевое слово FILE определяет собой своеобразный тип данных, а

указатель *fp указывает на этот тип.

Указатель файла указывает на структуру, содержащую различные сведения о файле, его имя, статус и указатель текущей позиции в начало файла.

Открытие файла осуществляется с помощью функции fopen(), которая открывает поток и связывает с этим потоком определенный файл. Прототип функции fopen() такой:

```
FILE  *fopen(const char *file_name,  const char *mode);
```

В прототипе функции fopen() формальные переменные имеют следующий смысл:

file_name – это имя файла с заданным расширением и возможным путем расположения, mode – режим работы файла: чтение, запись и т.д.

В табл. 8.2 приводятся допустимые значения режима для функции fopen().

Таблица 8.2.

Допустимые значения режима функции fopen()

№ п/п	Режим	Что означает
1.	**r**	Открыть текстовый файл для чтения
2.	**w**	Создать текстовый файл для записи
3.	**a**	Добавить в конец текстового файла
4.	**rb**	Открыть двоичный файл для чтения
5.	**wb**	Создать двоичный файл для записи
6.	**ab**	Добавить в конец двоичного файла
7.	**r+**	Открыть текстовый файл для чтения/записи
8.	**w+**	Создать текстовый файл для чтения/записи
9.	**a+**	Добавить в конец текстового файла или создать текстовый файл для чтения/записи
10.	**r+b**	Открыть двоичный файл для чтения/записи
11.	**w+b**	Создать двоичный файл для чтения/записи
12.	**a+b**	Добавить в конец двоичного файла или создать двоичный файл для чтения/записи

Например, для записи в файл с именем (и расширением) **data.txt** на диск **D** следует использовать такие объявление и операции:

```
FILE  *fp;
fp = fopen("D: \\data.txt", "w");
fprintf(fp, "\n\t hello, world\n");
fclose(fp);
```

В приведенном фрагменте **C++** – кода функция fclose() закрывает поток, который был открыт с помощью вызова функции fopen(). Функция fprintf() осуществляет форматную запись (в данном случае строку hello,

world) в файл. Все манипуляции с файлом происходят между функциями fopen() и fclose(). Режим функции fopen() задается строкой **"w"**, которая обеспечивает создание текстового файла для записи. Это означает, что файл **data.txt** создается на диске **D** и в него записывается строка hello, world с отступом от верхнего края и с отступом (табуляцией) от левого края.

Прототип функции fclose() следующий:

```
int fclose(FILE *fp);
```

В приведенной записи *fp – указатель файла, возвращенный в результате вызова функции fopen(). Возвращение нуля означает успешную операцию закрытия. В случае же ошибки возвращается EOF. Обычно отказ при выполнении функции fclose() происходит только тогда, когда диск был преждевременно удален из дисковода или на диске не осталось свободного места.

Правомочность открытия файла с помощью функции fopen() обычно подтверждается после проверки какой-либо ошибки, например, когда на диске нет места для записи или неправильного имени диска, причем эти ошибки будут обнаружены до того, как программа попытается в этот файл что-либо записать. Поэтому приведенный фрагмент **C++** – кода будет правильным, если производится проверка возможности открытия файла:

```
FILE  *fp;
 if ((fp = fopen("D:\\data.txt", "w")) == NULL)
{
//exit(1);
printf("\n\t Error! Can not open file\n ");
printf("\n Press any key: ");
_getch();
return 0;  }
fprintf(fp, "\n\t hello, world\n");
fclose(fp);
```

При выполнении условия проверки можно выходить при нажатии любой клавиши с заданным сообщением или немедленный выход сделать с помощью функции exit(), которая в данном фрагменте **C++** – кода закомментирована.

Функции для работы с текстовыми файлами удобно использовать при создании текстовых файлов, ведении файлов-протоколов и т.п. Но при создании баз данных целесообразно использовать функции для работы с бинарными файлами: fwrite() и fread(). Эти функции без каких-либо изменений копируют выделенный блок данных из оперативной памяти в файл и, соответственно, из файла – в память.

При записи или чтении суффикс "t" открывает файл в текстовом режиме. В этом режиме символ **CTRL+Z** (символ с кодом 26)

обрабатывается как символ конца файла. Кроме того, комбинации символов перевода строки и возврата каретки преобразуются в единственный символ перевода строки ('\n') при вводе, и символы перевода строки преобразуются в комбинации символов перевода строки и возврата каретки при выводе.

Суффикс "b" открывает файл в бинарном режиме, преобразования символов перевода строки и возврата каретки не производятся.

```
FILE  *fp;
  if ((fp = fopen("D:\\data.txt", "w")) == NULL)
{
          //exit(1);
 printf("\n\t Error! Can not open file\n ");
 printf("\n Press any key: ");
          _getch();  return -1;  }
fprintf(fp, "\n\t hello, world\n");
fclose(fp);
```

Практическая часть

Пример 1. *Напишите программу заполнения матрицы размера n × m нечетными целыми числами с выводом результата на консоль и в текстовый файл. Размеры матрицы и начальное нечетное число задаются пользователем с клавиатуры.*

Программный код решения примера:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int main(void) {
int i, j, x, xi, n, m, *matr;
FILE *fid;
char str[] = "D:\\data.txt"; // месторасположение файла
if ((fid = fopen(str, "w")) == NULL){
     printf("\n\t The file could not be opened.\n ");
     printf("\n Press any key: ");
_getch(); return 0; }

printf("\n\t Enter the number of lines: ");
scanf_s("%d", &n);
printf("\t Enter the number of columns: ");
scanf_s("%d", &m);
printf("\t Enter the odd number: "); scanf_s("%d", &x);
xi = x;
```

```
matr = (int *)calloc(n*m, sizeof(int));
// Заполнение матрицы целыми числами
for (i = 0; i < n; ++i)
for (j = 0; j < m; ++j)
{matr[i*m + j] = x; x += 2; }

printf("\n\t Matrix (%d x %d), initial number: %d\n",
n, m, xi);
fprintf(fid, "\r\n\t Matrix (%d x %d), initial number:
%d\r\n", n, m, xi);
for (i = 0; i < n; ++i){
     printf("\n "); fprintf(fid, "\r\n ");
     for (j = 0; j < m; ++j){
printf("%5d", matr[i*m + j]);
fprintf(fid, "%5d", matr[i*m + j]); }
}

fclose(fid);
printf("\n\n Result of record look in file %s\n", str);

printf("\n Press any key: ");
_getch();
return 0;
  }
```

В программу включена препроцессорная директива #define... для устранения предупреждения о ненадежной работе функции fopen() в Visual Studio 2008.

Возможный результат выполнения программы показан на рис. 8.1.

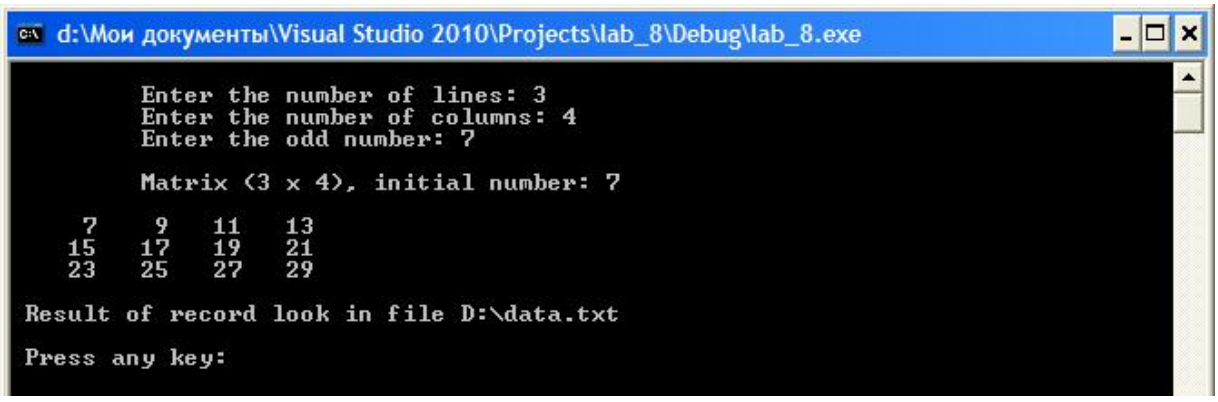


Рис. 8.1. Заполнение матрицы нечетными числами

Текстовый файл с заполненной матрицей показан на рис. 8.2.

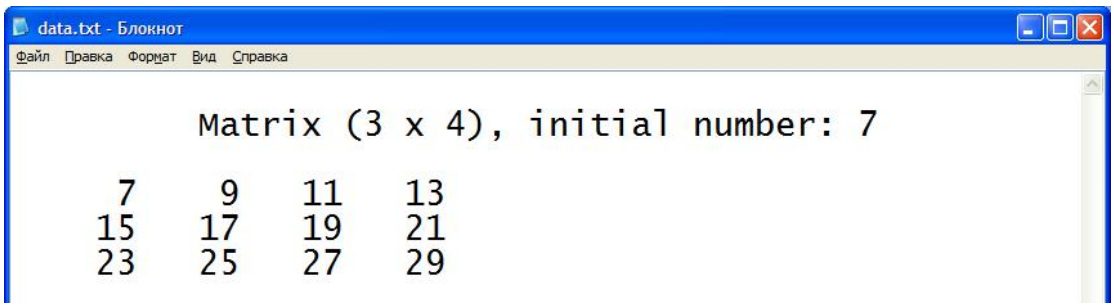


Рис. 8.2. Матрица нечетных чисел в текстовом файле

Примечание. В текстовом файле следует использовать моноширинный (равноширинный) шрифт, например, Courier New.

Пример 2. *Напишите программу форматированной записи в текстовый файл трех строк различной длины и одномерного целочисленного массива. Произведите чтение из текстового файла с выводом его содержания на консоль и преобразования одномерного массива в двухмерный.*

Для решения примера используем функции fprintf(), fgets(), atoi(), fscanf().

Программный код решения примера:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#define n 4 // Число строк матрицы
#define m 3 // Число столбцов матрицы
#define N 123 // Число считываемых строк из текстового
файла

int main(void) {
      int i, j = 0;
      int A[n*m] = {1,2,3,4,5,6,7,8,9,10,11,12};
      int B[n*m];
      FILE *fid;
    char *str[] = {"aza","baza","qwerty"};
    char str2[N][80]; // Буферный массив

      // Обнуление массива B[n*m]
      for (i = 0; i < n*m; ++i) B[i] = 0;

      if ((fid = fopen("D:\\data2.txt", "w")) == NULL)
```

```
{printf("\n\t The file could not be opened.\n ");
printf("\n Press any key: "); _getch(); return 0; }

// Запись в файл data2.txt
fprintf(fid, "\n\t The lines are:\n");
for (i = 0; i < m; ++i) fprintf(fid,"\t %s\n", str[i]);

 for (i = 0; i < n*m; ++i)
 fprintf(fid, " %3d", A[i]);

fclose(fid);

printf("\n\t From file \"data2.txt\":\n");
if ((fid = fopen("D:\\data2.txt", "r")) == NULL)
{printf("\n\t The file could not be opened.\n ");
printf("\n Press any key: "); _getch(); return 0; }

// Чтение из файла data2.txt
for (i = 0; (fgets(str2[i], 80, fid) != NULL) && (i <
N); ++i)
      printf("  %s",str2[i]);
fclose(fid);

if ((fid = fopen("D:\\data2.txt", "r")) == NULL)
{printf("\n\t Error! You can not open the file \n ");
printf("\n Press any key: "); _getch(); return 0; }
// Повторное чтение из файла data2.txt
for (i = 0; fscanf (fid, "%s", str2[i]) != EOF; ++i)
if (atoi(str2[i]))
{ B[j] = atoi(str2[i]); ++j; }

fclose(fid);

printf("\n\n\t The reconfigured array:\n");
for (i = 0; i < n; ++i) {
printf("\n\t");
for (j = 0; j < m; ++j)
printf("%5d", B[i*m+j]); }

printf("\n\n Press any key: ");
_getch();
 return 0;
 }
```

Для форматированной записи в текстовый файл и чтения из файла

применены массивы указателей *str[], str2[123][80]. Чтение из файла одномерного массива целых чисел выполняется с помощью функции atoi(), значения целых чисел заносятся сначала в одномерный массив B[n*m]. После закрытия файла **data2.txt** одномерный массив B[n*m] выводится на консоль в виде двухмерной матрицы размера 4 × 3. Форматированная запись строк и одномерного массива в файл **data2.txt** производится с помощью функции fprintf(). Первое чтение информации из текстового файла производится с помощью функции fgets(), что позволяет практически точно копировать расположение строк текстового файла на консоль (дисплей). Функция fscanf() используется для форматированного чтения информации из текста с последующим выделением целых чисел с помощью функции atoi().

Возможный результат выполнения программы показан на рис. 8.3.

Рис. 8.3. Содержимое текстового файла и преобразованного массива

Пример 3. *Напишите программу добавления слов в текстовый файл с контролем на консоли.*

В текстовый файл запишем название книги и авторов. После будем добавлять слова, символы и т.д.

Для программного решения примера используем функции файлового ввода/вывода fprintf(), fgets() и rewind(). Кроме того, подключим библиотеку **locale.h** и объявим прототип функции, что позволит использовать шрифты русского алфавита:>

```
#include <locale.h>
setlocale( LC_ALL, "Russian");
```

или

setlocale(LC_ALL, ".1251");//кодовая страница Windows–1251

Программный код решения примера:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <locale.h>
```

```
#define MAX 40

int main(void) {
           FILE *fid;
    char words[MAX+1];
char str_name[] = "D:\\data3.txt";
// Прототип функции поддержки русских шрифтов
     setlocale( LC_ALL, "Russian");

         if ((fid = fopen(str_name, "a+")) == NULL)
{fprintf(stdout, "\n\t Файл не может быть открыт
\"%s\".\n ", str_name);
  printf("\n Нажмите любую клавишу: ");
_getch(); return -1; }
printf("\n\t Введите слова для включения их в файл
\"%s\"\n\t\
и нажмите клавишу Enter в начале строки для завершения
ввода\n\t: ", str_name);
     // Запись в файл data3.txt
while (gets_s(words, MAX) != NULL && words[0] != '\0')
{printf("\t: "); fprintf(fid," %s\n", words); }

puts("\t Содержимое файла:");
// Устанавливает указатель текущей позиции в начало
файла
rewind(fid);

// Сканирование файла
while (fgets(words, MAX, fid) != '\0')
     printf("\t%s", words);

if (fclose(fid) != 0)
fprintf(stderr, "\n\t Ошибка при закрытии файла
\"%s\"\n", str_name);

printf("\n\n Нажмите любую клавишу (Press any key): ");
     _getch();
     return 0;
}
```

В программе введены две проверки: на открытие файла if (... == NULL) и на закрытие файла if (... != 0). Эти проверки позволяют исключить аварийный выход из программы. Использование в функции форматного вывода fprintf() ключевого слова stdout позволяет выводить сообщения на консоль – дисплей пользователя.

Вместо стандартной функции gets() использована функция gets_s(),

которую поддерживает MS Visual Studio. При работе в MS Visual Studio с функцией gets() появляются предупреждения (которыми в общем случае можно пренебречь). Предупреждения возникают и при работе с функцией fopen(). Вместо нее можно использовать fopen_s() в следующем формате записи:

```
fopen_s(&fid, "D:\\data3.txt","a+");
```

Тогда проверку на открытие файла следует изменить, например:

```
if (fopen_s(&fid, "D:\\data3.txt","a+"))
{fprintf(stdout, "\n\t Ошибка! Не удается открыть файл
\"data3.txt\".\n ");
printf("\n Нажмите любую клавишу: ");
_getch(); return -1; }
```

Если файл **data3.txt** сохранить, то при последующих выполнениях программы в этот файл будут дописывать данные. Это обеспечивает режим "a+" функции fopen().

Возможный результат выполнения программы показан на рис. 8.4.

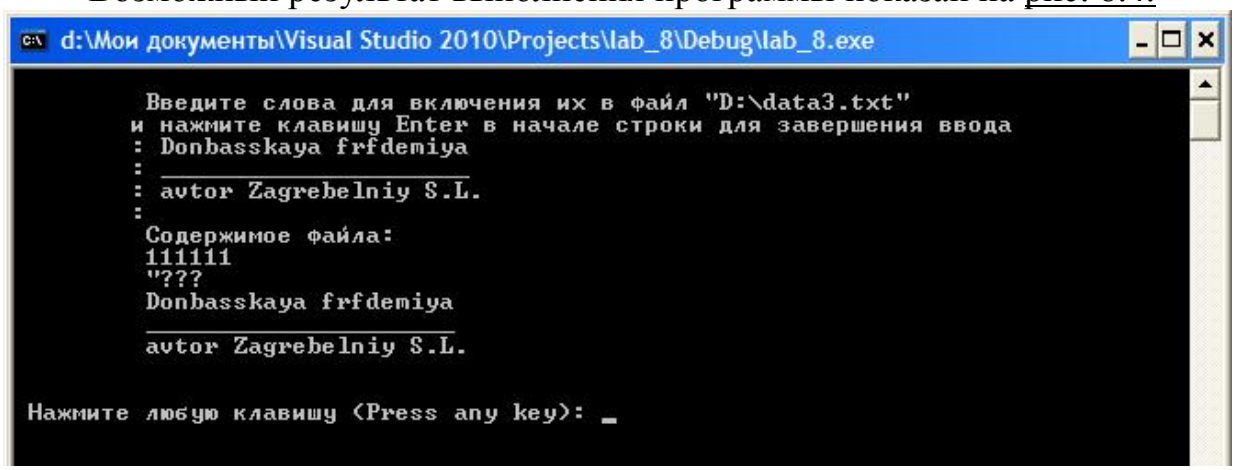


Рис. 8.4. Пример записи в файл и чтения из файла

Примечание. Для данной программы формат записи функции fscanf():

```
fscanf(fid, "%s", words);
```

Пример 4. *Напишите программу записи в файл нескольких строк и отображения содержимого файла в обратном порядке, как на консоли, так и в другом текстовом файле.*

Для решения примера используем функции fseek() и ftell().

Программный код решения примера:

```
#include <stdio.h>
```

```
#include <conio.h>
#define MAX 79
#define file "D:\\data6.txt" // запись в прямом порядке
#define file2 "D:\\data66.txt" // запись в обратном
порядке

int main(void) {
     char  ch, str[MAX+1];
     long n, m;
    FILE *fid, *fid2;

     if ( fopen_s(&fid, file, "w") ) {
fprintf(stdout, "\n\t The file could not be opened.\n
");
printf("\nPress any key: ");
_getch(); return 0; }

     printf("\n\t Enter a few lines and press Enter to
complete before the new line\n\t: ");
     // Запись в файл data6.txt
while (gets_s(str, MAX) != NULL && str[0] != '\0')
{ printf("\t: "); fprintf(fid," %s\n", str); }

fclose(fid);

if ( fopen_s(&fid, file, "r") ) {
fprintf(stdout, "\n\t File could not be opened.\n");
printf("\n Press any key: ");
_getch(); return 0; }

if ( fopen_s(&fid2, file2, "w") ) {
fprintf(stdout, "\n\t File could not be opened.\n");
printf("\n Press any key: ");
_getch(); return 0; }

//Переход в конец файла
fseek(fid, 0L, SEEK_END);
m = ftell(fid);

for (n = 1L; n <= m; n++) {
fseek(fid, -n, SEEK_END);
ch = getc(fid);

if (ch != '\n') {
     printf(" ");  putchar(ch);
```

```
fprintf(fid2, " "); putc(ch, fid2); }
} // End for
putchar('\n');

fclose(fid);
fprintf(fid2, "%c", '\n');
fclose(fid2);

printf("\n Result see the files, \"%s\" and \"%s\"\n",
file, file2);

printf("\n Press any key: ");
      _getch();
      return 0;
}
```

Функция fseek() имеет следующую форматную запись:

```
fseek(fid, 0L, SEEK_END);
```

Она определяет позицию со смещением в 0 байт от конца файла (именованная константа SEEK_END). Суффикс **L** означает тип long int.

Строка с функцией ftell() определяет количество байтов от начала до конца указанного файла. Это количество байтов записывается в переменную **m**:

```
m = ftell(fid);
```

Рассмотрим следующий программный цикл:

```
for (n = 1L; n <= m; n++) {
fseek(fid, -n, SEEK_END);
ch = getc(fid);
if (ch != '\n')
{ printf(" ");  putchar(ch);
fprintf(fid2, " "); putc(ch, fid2); }
} // End for
```

Первое выполнение цикла выводит программу на первый символ перед концом файла. Затем программа печатает этот символ на консоль и записывает в новый файл с именем **data66.txt**. Следующая итерация цикла выводит программу на предпоследний символ файла, который она печатает и записывает в новый файл. Этот процесс продолжается до тех пор, пока программа не выйдет на первый символ файла и не распечатает его (и запишет в файл).

Возможные результаты выполнения программы показаны на рис. 8.5-рис. 8.6 .

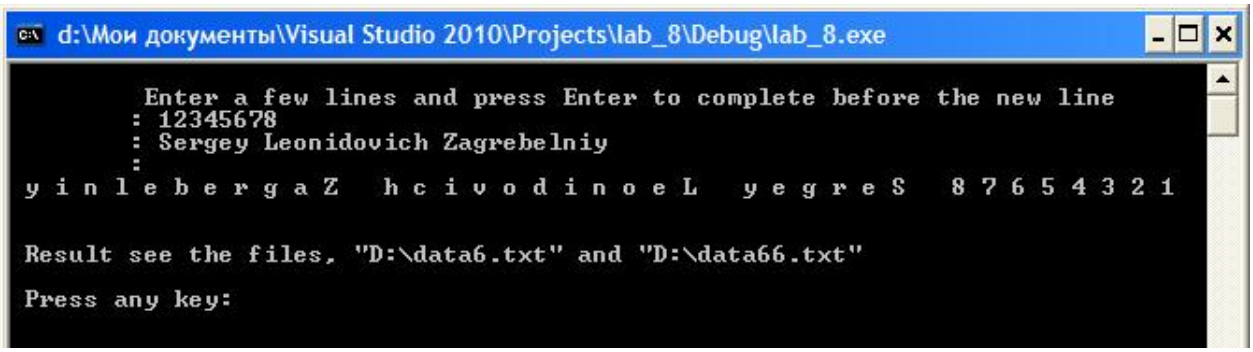


Рис. 8.5. Результат обратного считывания информации из файла

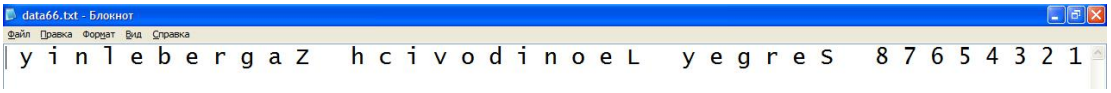


Рис. 8.6. Результат записи информации в файл в обратном порядке

Пример 5. *Создать на* ***СИ++*** *таблицу табулирования функции*

$$y=\begin{cases}2x+2, & \text{если } x\le 0\\ \sqrt{x+3}, & \text{если } 0<x\le a\\ \cos^2(x+2), & \text{если } x>a\end{cases}$$

с использованием оператора `While` *на отрезке* **[-2; 5]** *с шагом* **0,8.** *Результат записать в текстовый файл под именем «data_pr.txt». Произвести чтение данных из файла «data_pr.txt» и вывод их на консоль.*

Программный код решения примера:

```
// Восьмая программа на языке Си++
// Автор Загребельный С.Л.
#define _CRT_SECURE_NO_WARNINGS
#define MAX 79
#include <stdio.h>
#include <conio.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <limits.h>
#include <float.h>
#include <locale.h>

int main(void) {
```

```
        FILE *fid;
    double xn, xk, xh, a, x, y;
    int i;
char str_name[] = "D:\\data_pr.txt";
char words[MAX+1];
setlocale( LC_ALL, "Russian");
if ((fid = fopen(str_name, "w")) == NULL)
{fprintf(stdout, "\n\t Файл не может быть открыт
\"%s\".\n ", str_name);
  printf("\n Нажмите любую клавишу: ");
_getch(); return 0; }

printf("\n\t Введите xn= ");
scanf_s("%lf", &xn);
printf("\t Введите Xk= ");
scanf_s("%lf", &xk);
printf("\t Введите Xh= ");
scanf_s("%lf", &xh);
printf("\t Введите a= ");
scanf_s("%lf", &a);
x=xn;
while (x <= xk) {
    if (x<=0){ y=2*x+2;}else {
    if (x<=a) {y=sqrt(x+3);}else{
        if (x>a) {y=pow(cos(x+2),2);}}}
    fprintf(fid,"\n\t %4.3f\t\t%4.3f\n", x,y);
    x=x+xh;
}
fclose(fid);
printf("\n\t Чтение из файла \"data_pr.txt\":\n");
if ((fid = fopen("D:\\data_pr.txt", "r")) == NULL)
{printf("\n\t Файл не открывается.\n ");
printf("\n Нажмите любую клавишу: "); _getch(); return
0; }

puts("\t Содержимое файла:");
// Устанавливает указатель текущей позиции в начало
файла
rewind(fid);
// Сканирование файла
    while (fgets(words, MAX, fid) != '\0')
    printf("\t%s", words);

if (fclose(fid) != 0)
```

```
fprintf(stderr, "\n\t Ошибка при закрытии файла
\"%s\"\n", str_name);
fclose(fid);
printf("\n Press any key: ");
    _getch();
    return 0;
 }
```

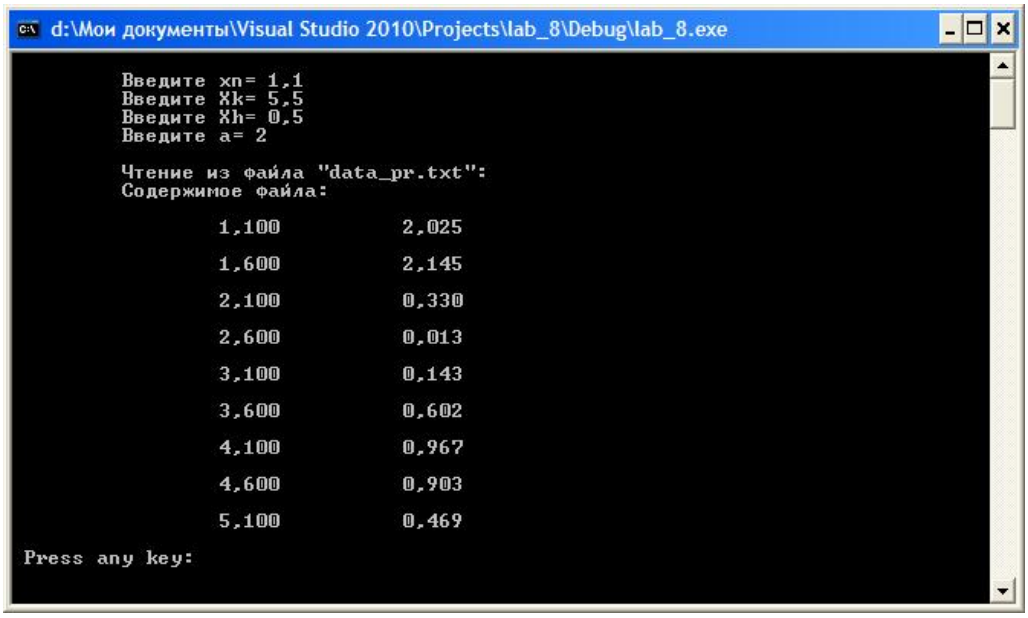


Рис. 8.7. Окно работы программы

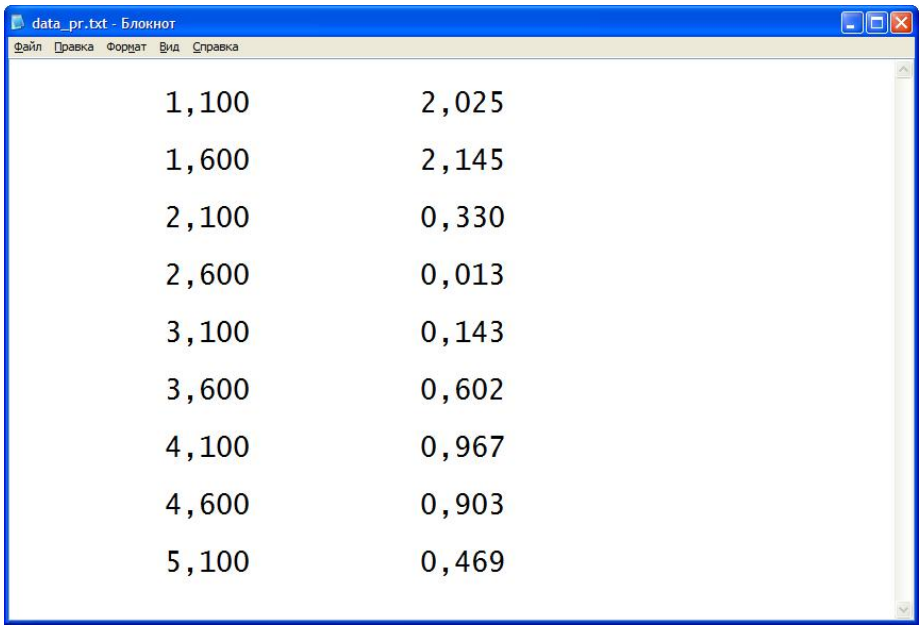


Рис. 8.8. Результат записи информации в текстовый файл «*data_pr.txt*»

Индивидуальные задания

Создать таблицу табулирования функции

$$y=\begin{cases} f1(x), & \text{если } x\le 0 \\ f2(x), & \text{если } 0<x\le a \\ f3(x), & \text{если } x>a \end{cases}$$

с использованием оператора цикла `For` *на отрезке* **[xn;xk]** *с шагом* **xh.** *Данные взять из таблицы 8.3. Результат табулирования записать в текстовый файл под именем* **«Tablica.txt»,** *а также произвести чтение данных из этого файла и сделать вывод на экран консоли.*

Таблица 8.3

Вариант	**Функции**			Границы отрезка **[xn;xk]**	Шаг табулирования, **xh**
	$f1(x)$	$f2(x)$	$f3(x)$		
1	$\ln\lvert x^2+5\rvert$	$\sin(e^x+2)$	$\dfrac{\sin(x+3)}{e^{2x}+\cos(x+1)}$	[-2,9; 8,2]	0,2
2	$2\sqrt{\lvert x^3\rvert}\sin(x^3)$	$(x+1)^2\cos x^3$	$\sqrt{x^4+2}+\sin x^2$	[-1,2; 2,6]	0,1
3	$\sin(x^5+3)$	$\sqrt{x^3}\sin x$	$x^4-\sin(x+1)$	[-1,7; 2,4]	0,3
4	$x^4\operatorname{tg}(x+2)$	$\ln(4x^2+1)$	$\ln\sqrt[5]{5+x^2}$	[-4,3; 8,0]	0,5
5	$x^5+\sqrt[3]{x+10}$	$1{,}3\sqrt{4+x^2}$	$\lvert x+1\rvert^x$	[-9,1; 5,8]	0,14
6	$\lvert x\rvert^5\operatorname{ctg}\lvert 2x\rvert$	$\ln(x^2+1)$	$e^{-2x}-\sqrt[3]{\lvert x+1\rvert}$	[-3,4; 2,5]	0,23
7	$x^5\operatorname{ctg}(2x^3)$	$\sqrt[5]{x^4+3}$	$\lvert\sin^2 x+1\rvert^{2x}$	[-2,2; 8,1]	0,15
8	$\operatorname{ctg}(3x-1)^2$	$2+xe^{-x}$	$\sin(x^3+1)$	[-2,8; 5,2]	0,5
9	$x^3+4x^2\sqrt{\lvert x\rvert}$	$(x-1)^3+\cos x^3$	$\sqrt{\lvert x\rvert^3}\sin x^3$	[-3,2; 7,8]	0,36
10	$(2x+1)(\lvert x\rvert+2)^3$	$e^x+\sin(x+2)$	$3\ln\sqrt[5]{\sin^2 x+2}$	[-6,1; 1,3]	0,15
11	$\operatorname{ctg}(x^3+1)$	$\ln(\sin x+1)^2$	$\sqrt[3]{2x^2+x^4+1}$	[-7,4; 0,6]	0,16
12	$1{,}3\sqrt{4+x^2}$	3^{x+3}	$5^{x+1}+\operatorname{tg}(x+1)$	[-1,2; 7,1]	0,45
13	$e^{2x}+\sin(2x^3)$	$\sin^3 x^4$	$e^{-x}+\sqrt[3]{3x^2+1}$	[-2,2; 3,9]	0,55
14	$x^3+(\lvert x\rvert+1)^{0,1x}$	$(x-1)^3+\cos(x^3)$	$2x+\operatorname{tg}(x^2+2)$	[-0,3; 4,5]	0,62

Продолжение таблицы 8.3.

15	$\left\|\sqrt[3]{\frac{2x+5}{(x^3+2)}}\right\|$	$\frac{5x+x^2}{(x^2+3)^3}$	$\cos^2(x^3+\sqrt{x})$	[-2,4; 4,4]	0,4
16	$\sqrt[5]{x^2+x+1}$	$\ln^2(\left\|\sqrt{x+5}\right\|)$	$\sin(x^2)+x^{0,25}$	[-3,9; 3,8]	0,15
17	$3x^5+\mathrm{ctg}(x^3+1)$	$e^{x+1}-\sin(\pi x)$	$\sqrt[5]{\sin^2 x+2}$	[-1,3; 7,1]	0,6
18	$x^5-\mathrm{ctg}(\pi x^3)$	$(\left\|7x\right\|+1)^{0,3}+\sin x$	$5x-x^2$	[-2,9; 6,2]	0,8
19	$\left\|x\right\|^{x+2}+\sin(x)$	$3^{x+3}+2x$	$\sqrt[5]{x^2+x+1}$	[-3,7; 8,5]	0,11
20	$x^2+\sin(7x)-1$	$\left\|x^3+10^x\right\|$	$\sqrt[7]{2x^4+x^2+1}$	[-3,9; 1,2]	0,25
21	$\sqrt[3]{\left\|x\right\|+2}-1$	$\sin(x^2)+x^{0,25}$	$\ln^2(x)+\sqrt{x}$	[-4,5; 6,1]	0,3
22	$\sqrt{\left\|\sin^2 x+\cos^4 x\right\|}$	$\ln(x+1)+\sqrt{3x}$	$5^{x+1}+\mathrm{tg}(x+3)^2$	[-3,4; 3,4]	0,33
23	$x^3-3x^2\sqrt{\left\|x\right\|+6}$	$\frac{2x+2}{\mathrm{tg}(2x-1)+1}$	x^4-x^x	[-4,1; 5,0]	0,45
24	x^4+2x^3-x	$e^{-x}+\sqrt[3]{x}$	$\ln(\left\|x^3+x^2\right\|)$	[-1,7; 2,9]	0,75
25	$\frac{(3x-1)^2}{x^5+2x+1}$	$\ln^2\left\|\sqrt{x+5}\right\|$	$\sqrt[5]{1+x^2}$	[-1,6; 4,7]	0,65
26	$x^5\mathrm{ctg}(2x^3)$	$\frac{5}{\mathrm{tg}(2x+3)+1}$	x^2e^{-x}	[-1,6;3,7]	0,3
27	$x^x\mathrm{tg}(x+5)$	$x^3\cos x$	$\sin x^2+x^{0.25}$	[-2,8; 8,2]	0,4
28	$\ln\left\|x+1\right\|+\sqrt{3\left\|x\right\|}$	$\sin(x^2+3x)$	$\ln^2\left\|x\right\|+\sqrt{x+3}$	[-1,7; 2,6]	0,25
29	$\sin^2 x^3$	$\sqrt[5]{6x-x^2+1}$	$\sin(x-e^{-x})$	[-2,2; 7,4]	0,23
30	$\cos(x^3+1)e^{-x}$	$\left\|\sqrt[3]{\frac{2x+5}{(x^3+2)}}\right\|$	$2\sqrt{x^2+7\sin(x^3)}$	[-1,1; 7,9]	0,8

Контрольные вопросы

1. Что может быть файлом в языке **C++**?
2. Какие обязательные операции выполняются при нормальной работе

с файлами? Какие библиотечные функции при этом используются?

3. Как определяется текстовой поток в стандарте языка **C++**?
4. Как определяется двоичный поток в стандарте языка **C++**?
5. Что определяет собой указатель файла?
6. С помощью каких функций языка **C++** осуществляется форматная запись в файл и форматное чтение данных из файла?
7. Какая переменная стандартной библиотеки используется для определения стандартного потока вывода на дисплей?
8. Какая переменная стандартной библиотеки используется для определения стандартного потока чтения с дисплея?
9. Как в языке **C++** кодируется признак конца файла?
10. Как в языке **C++** кодируется признак конца строки?
11. Что такое файл произвольного доступа?
12. Как в языке **C++** осуществляется пакетная запись данных в файл?
13. Как осуществляется запись бинарной информации в текстовый файл?
14. Как осуществляется чтение бинарной информации из текстового файла?

САМОСТОЯТЕЛЬНАЯ РАБОТА

Обработка элементов диагоналей квадратных матриц

Теоретическая часть

Для доступа к элементу массива следует указать имя массива с последующим числом (индексом), заключенном в квадратные скобки.

Элементы массива можно использовать в любом выражении точно также как и значение константы или переменной.

Например:

```
a[0][0]=11.2;
a[1][2]=10.2;
a[3][1]=22.1;
a[4][2]=1.1;
Y = 2*a[0][1] – a[1][0];
```

Селективная обработка массива – это выделение из массивов элементов, удовлетворяющих условию, и обработка выделенных фрагментов. Часто из выделенных фрагментов формируют новый (рабочий) массив, который далее и обрабатывают.

Наиболее часто встречаются такие условия обработки элементов массива:

–	четные	A[i] %2 ==0
–	нечетные	A[i] %2 !=0
–	кратные k	A[i] % k ==0
–	не кратные k	A[i] % k !=0
–	стоящие на четных местах	i % 2==0
–	стоящие на нечетных местах	i % 2!=0
–	положительные	A[i] >0
–	отрицательные	A[i] <0
–	в интервале (x1,x2)	(A[i]>x1) && (A[i]<x2)

При обработке двумерных массивов возникает необходимость вычисления суммы, произведения, количества, среднего арифметического, максимума, минимума элементов каждой строки или каждого столбца, заданной строки или заданного столбца. В таком случае, при обработке массивов нужна организация вложенных циклов.

Цикл, который содержит другой цикл называют *внешним* циклом, а цикл, содержащийся в теле другого цикла, называют *внутренним*. Все операторы внутреннего цикла должны полностью располагаться в теле внешнего цикла.

Поэтому, если за счетчик внешнего цикла взять индекс строки, а за счетчик внутреннего – номер столбца, то обработка двухмерного массива будет идти по строкам, а если наоборот, то по столбцам.

При обработке двумерных массивов часто приходится выделять элементы:

– *k – й строки*	*A[i][j], где i==k, j=1, …, M*
– *k – го столбца*	*A[i][j], где i=1, …, N; j==k*

а для квадратных матриц (M=N) также:

– *главной диагонали*	*A[i][i], где i=1, …, N*
– *побочной диагонали*	*A[i][N+1-i], где i=1, …, N*
– *наддиагональные*	*A[i][j], где i>j*
– *поддиагональные*	*A[i][j], где i<j*

Существует множество алгоритмов для сортировки массивов. Ниже рассмотрены два из них: *сортировка выбором* и *методом пузырька.*

1. Сортировка выбором

Суть этого метода очень проста и может быть описана так:

1. В последовательности из *n* элементов выбирается наименьший (наибольший) элемент;
2. Меняется местом с первым;
3. Далее процесс повторяется с оставшимися *n*-1 элементами, затем с оставшимися *n*-2 элементами и т.д., до тех пор пока не останется один самый большой (маленький) элемент.

Для реализации этого алгоритма необходимо использовать два вложенных цикла с параметром `For`. Внешний цикл (по `i`) предназначен для последовательного фиксирования элементов массива, внутренний (по `j`) - осуществляет поиск минимального (максимального) и его позиции в неотсортированной части массива. После выхода из внутреннего цикла следует перестановка элементов. Последний элемент во внешнем цикле не рассматривается: он сам встанет на свое место.

2. Сортировка методом пузырька

Метод основан на сравнении соседних элементов. «Неправильно» расположенные по отношению друг к другу элементы меняются местами. Во вложенных циклах поочередно фиксируется пара соседних элементов массива. В результате первого прохода элемент с минимальным значением оказывается в первой позиции массива (всплывает).

Уплотнение массива – это удаление из него элементов, отвечающих

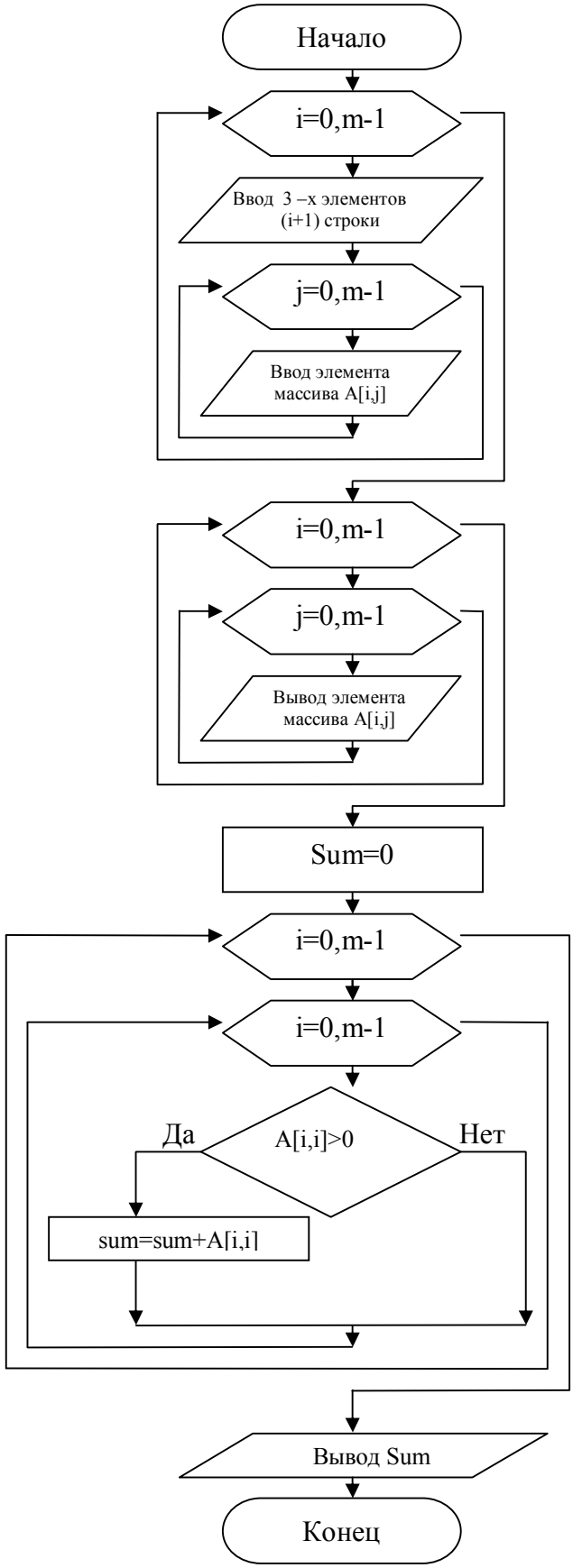
тем или иным условиям. Образующиеся пустоты заполняются за счет сдвига всех оставшихся элементов. Так как массив укорачивается, при обработке массива необходимо использовать не цикл с параметром, а цикл с условием.

Вставка элемента в массив – задача обратная предыдущей. Прием используется тот же – смещение группы элементов на одну позицию. Только при уплотнении сдвиг производится влево, при вставке – вправо. При вставке возникает проблема, что делать с последними элементами? Если в дальнейшей работе с массивом участвуют только заявленные элементы, то «хвост» придется вытеснить, последние значения при этом будут утрачены. Иначе, нужно создавать дополнительный массив, размерность которого будет больше исходного на количество вставленных элементов. Выбор типа цикла для работы с массивом зависит от конкретного случая.

Практическая часть.

Пример 1. *Создать блок-схему к заданию и программу на СИ++ для нахождения суммы положительных элементов главной диагонали. (Матрица квадратная, ввод элементов сделать с клавиатуры).*

Блок-схема к программе



Программный код решения примера:

```
//Девятая программа
//Автор Загребельный С.Л.
#include <stdio.h>
#include <conio.h>
#include <locale.h>
#define m 3
int main (void) {
    int i, j, k, l,sum; // переменные циклов
int A[m][m];
setlocale(LC_ALL, "Russian");
    // Ввод элементов матрицы
    for (i = 0; i < m; i++)
    {printf("\n Введите 3 элемента %d строки\n",i+1);
        for (j = 0; j < m; j++)
            scanf("%i,%j",&A[i][j]);}
// Распечатка  матриц
    printf("\n  Исходная матрица (%dx%d):\n", m, m);
    for (i = 0; i < m; i++) {
        printf("\n");
        for (j = 0; j < m; j++)
            printf(" %4d", A[i][j]);
    }
sum=0;
for (i = 0; i < m; i++)
        for (i = 0; i < m; i++)
            if (A[i][i]>0) {sum = sum+A[i][i];}
printf(" \n сумма положительных элементов главной
диагонали %d\n", sum);
    printf("\n\n ... Press any key: ");
    _getch();
    return 0;
}
```

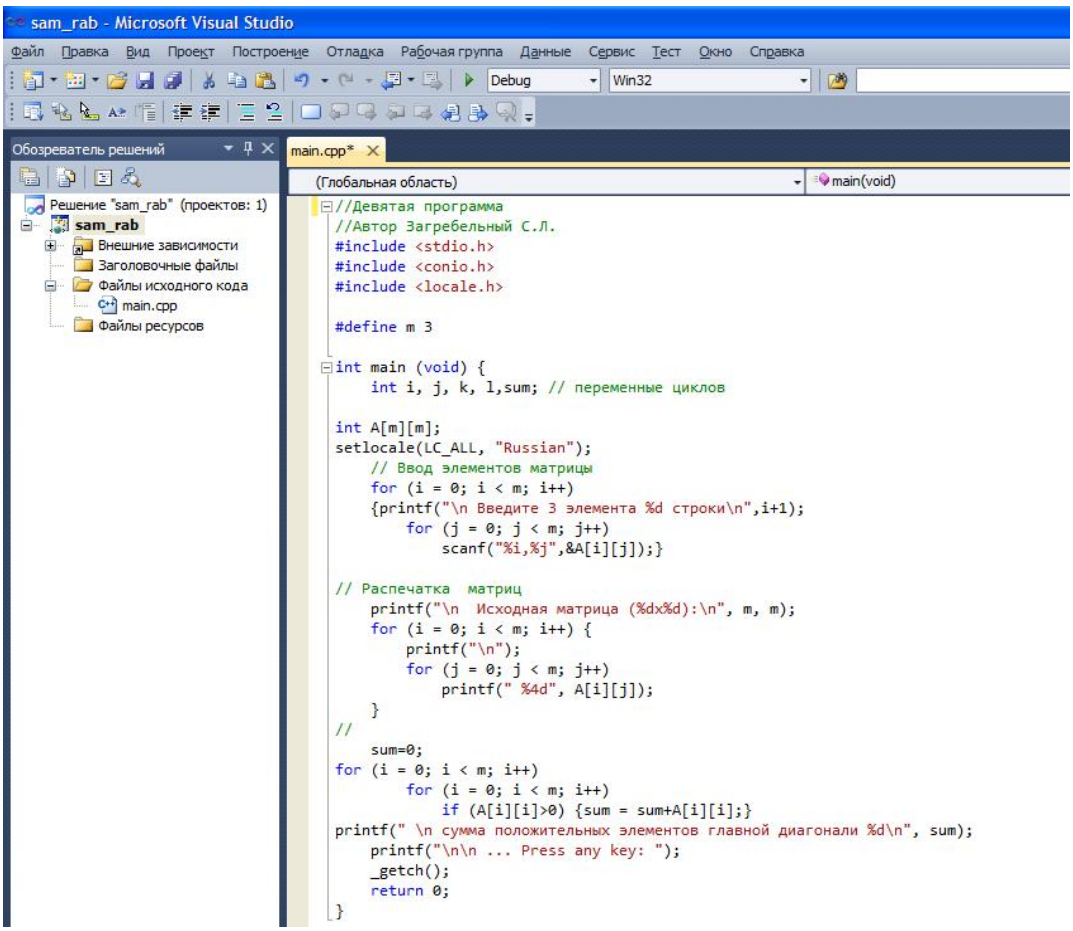


Рис. 9.1. Окно кода программы

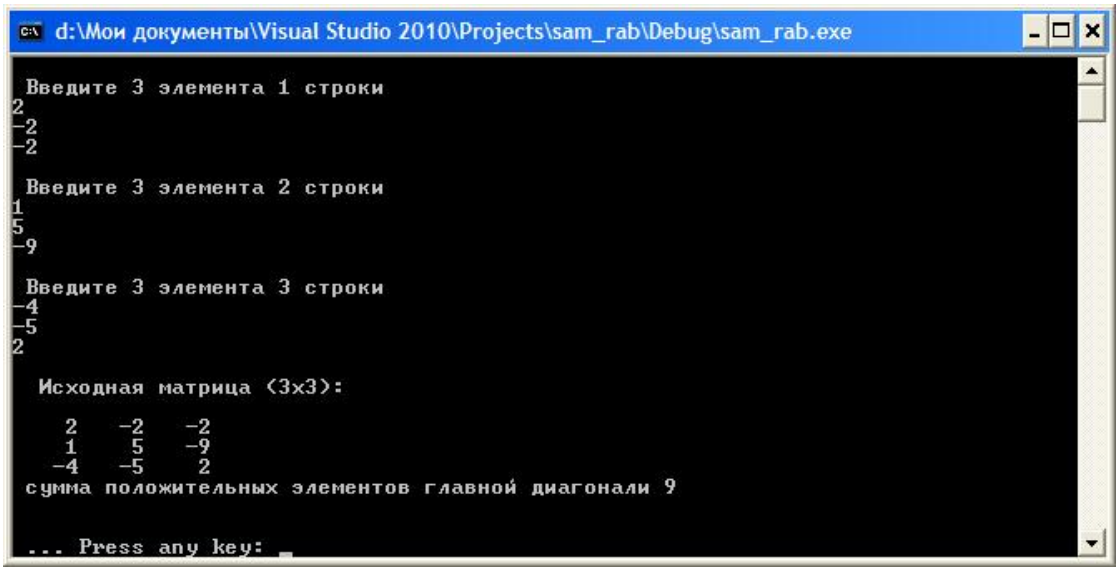


Рис.9.2. Окно выполнения программы

Индивидуальные задания

Составить блок-схему к заданию и программу на языке программирования С++ для решения задания из таблицы 9.1.(Матрица квадратная размером N×N, ввод элементов массива сделать автоматически через генератор случайных чисел, элементы массива должны быть целыми числами).

Таблица 9.1.

Номер варианта	Условие задания
1	Общую сумму положительных четных чисел на главной и побочной диагоналях.
2	Общее произведение нечетных отрицательных элементов главной и побочной диагоналей.
3	Количества кратных 3 элементов отдельно на главной и отдельно на побочной диагоналях.
4	Среднее арифметическое для отрицательных элементов главной диагонали и среднее арифметическое для положительных элементов побочной диагонали.
5	Найти среднее геометрическое положительных кратных 4 элементов главной и побочной диагоналей.
6	Где больше кратных 3 элементов: на главной или побочной диагоналях.
7	Где меньше отрицательных элементов: на главной или побочной диагоналях.
8	Что больше: произведение положительных элементов главной диагонали или произведение отрицательных побочной.
9	Что меньше: сумма нечетных элементов главной диагонали или произведение некратных 3 элементов побочной.
10	Общую сумму отрицательных нечетных чисел на главной и побочной диагоналях.
11	Общее произведение четных кратных 3 элементов главной и побочной диагоналей.
12	Количества кратных 5 элементов отдельно на главной и отдельно на побочной диагоналях.
13	Суммы положительных четных чисел отдельно на главной и отдельно на побочной диагоналях.
14	Произведения нечетных отрицательных элементов отдельно на главной и отдельно на побочной диагоналях.
15	Общее количество кратных 3 элементов на главной и побочной диагоналях.

Продолжение таблицы 9.1.

16	Среднее арифметическое для всех отрицательных элементов главной и побочной диагоналей.
17	Найти среднее геометрическое положительных элементов отдельно для главной и отдельно для побочной диагоналей.
18	Где больше сумма кратных 4 элементов: на главной или побочной диагоналях.
19	Где меньше сумма положительных элементов: на главной или побочной диагоналях.
20	Что больше: сумма четных элементов главной диагонали или сумма нечетных побочной.
21	Что меньше: произведение кратных 4 элементов главной диагонали или сумма положительных побочной.
22	Суммы отрицательных нечетных чисел отдельно на главной и отдельно на побочной диагоналях.
23	Произведения четных элементов отдельно на главной и отдельно на побочной диагоналях.
24	Общее количество не кратных 5 элементов на главной и побочной диагоналях.
25	Общую сумму квадратов положительных элементов главной и побочной диагоналей.
26	Среднее арифметическое нечетных элементов главной диагонали.
27	Произведение суммы четных элементов главной диагонали на количество нечетных элементов побочной диагонали.
28	Количество элементов кратных числу 3 из диапазона [-10;10] для главной диагонали.
29	Среднее геометрическое нечетных элементов главной и побочной диагонали.
30	Сумму нечетных отрицательных элементов главной диагонали и произведение четных чисел из диапазона [-5;6] для побочной диагонали.

Контрольные вопросы

1. Понятие матрицы, ввод элементов матрицы с клавиатуры (написать фрагмент программы).
2. Селективная обработка элементов массива (четность, нечетность и т.д.).
3. Различие сортировки методом пузырка и методом выбора, какой из них быстрее.
4. Какое условие отбора элементов главной и побочной диагонали, наддиагональных элементов.