МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

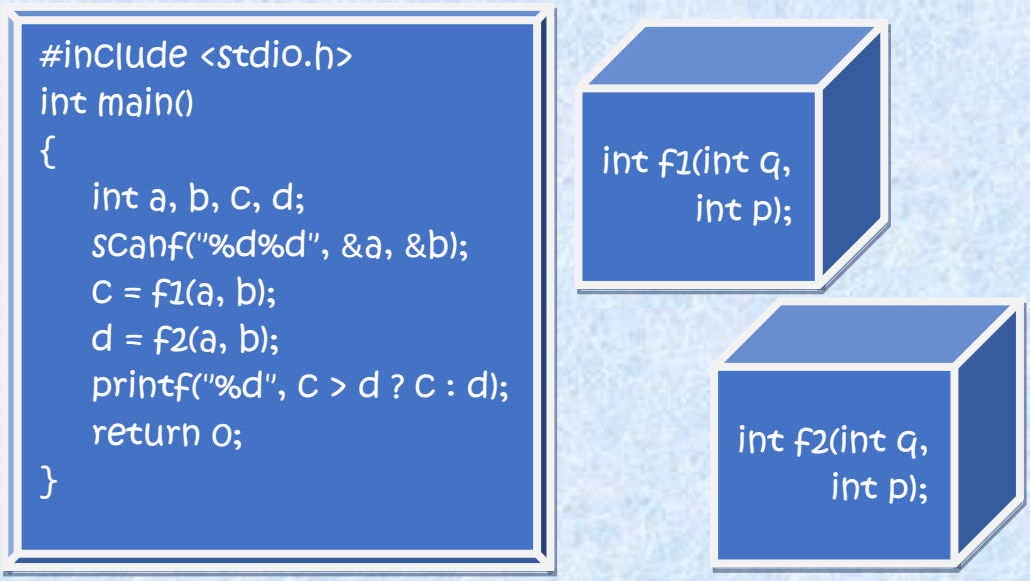
М.Г. ЖУРАВЛЕВА, В.А. АЛЕКСЕЕВ, П.А. ДОМАШНЕВ

ОСНОВЫ ПРОГРАММИРОВАНИЯ. ВВЕДЕНИЕ В ЯЗЫК СИ. ЧАСТЬ 1

1. УЧЕБНОЕ ПОСОБИЕ

по курсам «Программирование»,

«Основы алгоритмизации и программирования»



Липецк
Липецкий государственный технический университет
2019

УДК 004.43 (07)
Ж911

Рецензенты

кафедра информатики, информационных технологий и защиты информации ЛГПУ имени П.П.Семенова-Тян-Шанского;

В.И. Сумин, проф. кафедры информационной безопасности телекоммуникационных систем Воронежского института ФСИН России,
д-р техн. наук, проф.

Авторский коллектив: Журавлева М.Г. – предисловие, введение, гл. 2-3, Алексеев В.А. – гл. 1, Домашнев П.А. – гл. 4.

Журавлева, М.Г.

Ж911 Основы программирования. Введение в язык Си. Часть 1 : учебное пособие по курсам «Программирование», «Основы алгоритмизации и программирования» / М.Г. Журавлева, В.А. Алексеев, П.А. Домашнев. – Липецк: Изд-во Липецкого государственного технического университета, 2019. – 99 с. – Текст: непосредственный.

ISBN 5-00175-000-0

ISBN 5-00175-001-7 (Ч. 1)

Учебное пособие предназначено для изучения программирования, в том числе, дисциплин «Программирование», «Основы алгоритмизации и программирования» обучающимися всех направлений и специальностей высшего образования и среднего профессионального образования. Содержит введение в архитектуру компьютера, алгоритмизацию и программирование на языке Си. Рассмотрены лексические элементы, операции, операторы языка Си, ввод/вывод, базовая обработка массивов в циклах, вопросы создания и трансляции программ и др. По каждой теме представлены контрольные вопросы и упражнения.

Табл. 7. Ил. 3. Библиогр.: 5 назв. УДК 004.43 (07)

Печатается по решению редакционно-издательского совета ЛГТУ

ISBN 5-00175-000-0
ISBN 5-00175-001-7 (Ч. 1)

ОГЛАВЛЕНИЕ

Предисловие

Учебное пособие предназначено для обучающихся образовательных учреждений высшего образования по направлениям и специальностям высшего образования и среднего профессионального образования, образовательные программы которых включают курс программирования, и всех тех, кто желает самостоятельно освоить основы алгоритмизации и программирования.

Дисциплины «Программирование», «Основы алгоритмизации и программирования» изучаются в течение двух семестров и являются ключевыми в соответствующих образовательных программах как с точки зрения целей получения обучающимися знаний, умений и навыков, необходимых в будущей профессиональной деятельности, так и в качестве основы для большей части дисциплин, изучаемых позднее.

Материал пособия предназначен для изучения обучающимися основ алгоритмизации, элементов языка Си и для овладения начальными навыками программирования на языке Си с использованием вычислительных систем. В пособии подробно описаны отдельные лексические элементы, операции, операторы языка Си, уделено внимание вводу/выводу, обработке одномерных и двумерных массивов в циклах, вопросам трансляции, создания и запуска программ. Для закрепления знаний в пособие включены контрольные вопросы и упражнения, подробно описаны те моменты, которые вызывают сложности у начинающих программистов.

В качестве инструментального средства для написания программ выбран процедурный язык программирования Си (C) – основа многих современных языков и платформ разработки программного обеспечения. Для освоения предмета следует установить одну из доступных сред для разработки программ на языках C/C++, например, Code Blocks, Microsoft Visual Studio.Net, или использовать онлайн-компилятор.

Введение

Программированием для вычислительных систем (ВС) в широком смысле называют реализацию процессов разработки алгоритма, создания, трансляции, исполнения, тестирования, отладки, ввода в эксплуатацию и сопровождения программы для ВС с целью решения некоторой заранее поставленной задачи. В узком смысле под программированием понимают разработку программы для решения задачи с использованием средств автоматизации вычислительных процессов. Программа состоит из последовательности операторов используемого языка программирования и выполняется некоторым исполнителем, в частности, центральным процессором ВС. Оператор инициирует выполнение одного или нескольких действий исполнителем. Список всех возможных действий (команд) определяется для исполнителя заранее, на этапе проектирования ВС.

Если говорить о программировании, не ограничиваясь рамками средств автоматизации вычислений, можно утверждать, что жизнь человека состоит из множества программ: завтрак, поход на работу или учебу, отдых в парке, катание на лыжах. Эти и другие процессы состоят из действий, которые выполняются в том или ином порядке. Несмотря на то, что самих действий может быть очень много, количество групп однотипных действий ограничено. В частности, можно выделить такие группы:

1. Выполнить что-либо однократно: съесть кусочек торта.

2. Выполнять что-либо многократно: проехать 10 кругов на лыжах в парке.

3. Выполнять разные действия в зависимости от результата выполнения какого-либо условия: если светофор показывает красный свет, ожидать; если светофор показывает желтый свет, приготовиться; если светофор показывает зеленый свет, перейти дорогу.

4. Перейти к выполнению последующих действий, пропустив выполнение некоторой последовательности действий: лечь спать, не выполнив домашнее задание.

5. Попросить кого-либо выполнить определенные действия: заказать пиццу с доставкой на дом.

Между перечисленными группами действий и группами операторов языков программирования есть много общего.

Задача и метод ее решения, алгоритм, программа, вычислительная машина – таков краткий список понятий, относящихся к программированию. Задача поставлена с целью получения определенного результата и характеризуется совокупностью объектов. Некоторые из них являются заданными и называются входными данными, другие – подлежат определению и называются выходными данными. Одна и та же задача может быть решена несколькими различными методами. Метод задает совокупность действий, позволяющих решить задачу, т.е. преобразовать входные данные в выходные. Алгоритм формализует и детализирует метод, разбивая его на определенную, конечную последовательность подзадач или шагов, которую нужно выполнить для получения результата. Для автоматического выполнения алгоритма нужен исполнитель, являющийся частью вычислительной машины или ВС.

Вычислительная машина может быть реальным физическим устройством, состоящим из микросхем, проводов и т.п., т.е. аппаратной вычислительной машиной, или может быть построена посредством программ, выполняемых на другой вычислительной машине, т.е. программно-моделируемой вычислительной машиной [1]. Машина, состоящая частично из аппаратуры, а частично из программного обеспечения в [1] названа виртуальной вычислительной машиной. К последнему типу вычислительных машин относится ВС (или, что то же – компьютер) – совокупность аппаратно-программных средств, реализующих процессы обработки информации. Вычислительные машины такого типа в настоящее время широко распространены, поэтому в данном пособии речь пойдет о выполнении программ в ВС.

Если в качестве исполнителя применяется вычислительное устройство ВС, необходимо перевести алгоритм на язык, понятный этому устройству. С

этой целью выполняется перевод шагов алгоритма в операторы языка программирования, т.е. написание программы или кодирование – получение программного кода для ВС. Программный код состоит из последовательности команд (операторов). Таким образом, программа представляет собой дальнейшую детализацию алгоритма, выполненную в терминах используемого языка программирования. Если программа написана на одном из языков высокого уровня, то команды, из которых она состоит, слишком сложны и всё еще не могут быть выполнены исполнителем ВС. Поэтому формализация исходной задачи должна быть продолжена. В таком случае необходим перевод программы на язык машинных команд, известный исполнителю ВС.

1. Основные сведения об алгоритмизации, архитектуре ВС и средствах разработки программ

1.1. Об алгоритмах и алгоритмизации

В IX веке термином «алгоризм», впоследствии замененным на «алгоритм», были названы правила выполнения четырех арифметических действий, по имени сформулировавшего их узбекского математика Аль-Хорезми. В настоящее время под алгоритмом понимают точное предписание о выполнении в определенном порядке конечной совокупности операций для решения некоторого класса задач. Чтобы решить задачу, нужно определить последовательность шагов, которая приведет к получению верного результата, т.е. выполнить алгоритмизацию. Алгоритм должен обладать следующими основными свойствами:

- конечность и дискретность – алгоритм состоит из конечной последовательности отдельных простых (дискретных) шагов, должен завершаться и выдавать результат;

- определенность – состояние системы в любой момент времени однозначно определяет следующий шаг алгоритма;
- результативность – возможность получения требуемых результатов после завершения;
- универсальность – возможность применения алгоритма для различных наборов входных данных.

Следует заметить, что традиционный исполнитель ВС, т.е. устройство, которое выполняет команды программы, совсем не похоже на человеческий мозг. Исполнитель распознает команды из своей системы команд, заданной для конкретной ВС. Когда говорится, например, о задаче поиска минимального из двух чисел, подразумевается, что такой исполнитель не может «посмотреть» на эти два числа и сразу выдать результат. Однако он может выполнить команду сравнения этих чисел, а затем – команду вывода одного из чисел (меньшего) на внешнее устройство в качестве результата. Алгоритм должен также предусматривать ввод данных в память ВС и может выглядеть следующим образом:

Шаг 1. Занести первое число в ячейку памяти A1.

Шаг 2. Занести второе число в ячейку памяти A2.

Шаг 3. Сравнить содержимое ячейки памяти A1 с содержимым ячейки памяти A2 с помощью операции «<».

Шаг 4. Если результат операции – истина, т.е. содержимое ячейки A1 меньше содержимого ячейки A2, вывести на внешнее устройство содержимое ячейки A1 и закончить алгоритм. В противном случае – перейти на шаг 5.

Шаг 5. Сравнить содержимое ячейки памяти A1 с содержимым ячейки памяти A2 с помощью операции «>».

Шаг 6. Если результат операции – истина, вывести на внешнее устройство содержимое ячейки A2 и закончить алгоритм. В противном случае – перейти на шаг 7.

Шаг 7. Вывести на внешнее устройство текстовое сообщение о том, что введенные числа равны и среди них нет минимума, и закончить алгоритм.

Данный алгоритм вначале проверяет, меньше ли первое число, чем второе, если да – выводит меньшее и заканчивает работу. Если же первое число не меньше второго, то это означает, что либо второе число меньше первого, либо они равны – в этом случае минимума не существует. Соответствующие действия отражены в шагах 5-7 алгоритма.

Полезно рассмотреть традиционный алгоритм поиска минимального числа из множества n чисел. После загрузки всех чисел в память (каждое число помещается в отдельную ячейку) одно из чисел запоминается в некоторой дополнительной ячейке памяти. Далее содержимое этой ячейки последовательно сравнивается с оставшимися числами. Если какое-либо из оставшихся чисел меньше, чем число, которое находится в дополнительной ячейке, содержимое дополнительной ячейки обновляется – в нее записывается это меньшее число. Таким образом, алгоритм последовательно сравнивает первое число со вторым, результат первого сравнения (т.е. минимальное из первых двух чисел) с третьим числом, результат второго сравнения с четвертым числом и т.д. до конца, включая сравнение результата (n-1)-го сравнения с n-м числом. После окончания сравнений содержимое дополнительной ячейки выводится пользователю в качестве результата работы алгоритма.

Словесное описание алгоритма является одним из способов его записи, но не самым эффективным, так как не всегда бывает настолько детальным, чтобы сразу перевести его в программный код. В частности, упомянутый выше пошаговый алгоритм поиска минимума из двух чисел достаточно детален, чего нельзя сказать о следующем далее словесном описании алгоритма поиска минимума из множества чисел. В целом словесное описание часто применяется для раскрытия идеи алгоритма. Для промежуточной детализации, осуществляемой с целью дальнейшего перевода алгоритма в программный код, а также для описания самой программы можно использовать графические схемы (блок-схемы, диаграммы Насси-Шнайдермана и др.) или псевдокод [2].

Графическая схема содержит графические элементы, соответствующие шагам алгоритма, которые связываются друг с другом в определенном порядке.

Элементы блок-схем описаны в действующем ГОСТ 19.701-90[1] и представляют собой символы (блоки), отражающие действия или данные, а также линии, соединения и другие специальные символы. Для составления блок-схем алгоритмов и программ удобно использовать символы, представленные на рис. 1:

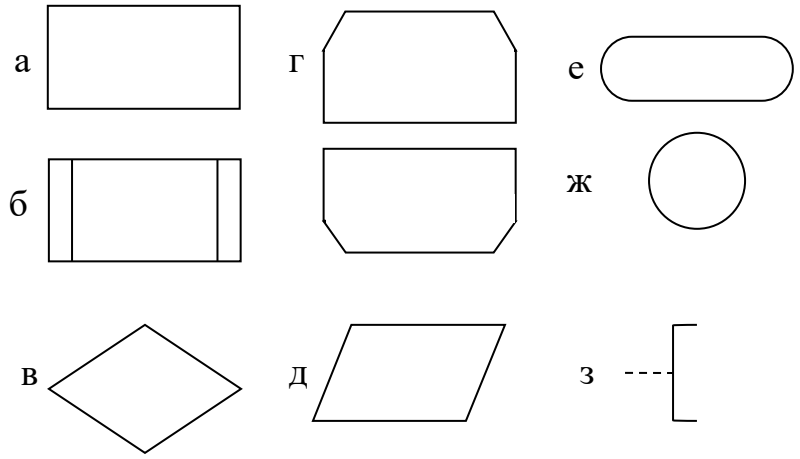


Рис. 1. Некоторые символы блок-схемы

а – символ процесса; отображает выполнение одной или нескольких операций;

б – символ предопределенного процесса; отображает одну или несколько операций, которые определены в другом месте;

в – символ решения; отображает функцию, переводящую один вход в один из множества выходов после проверки условия, содержащегося внутри символа;

г – символ границы цикла[2], состоит из двух частей; отображает начало и конец цикла; если условие завершения цикла проверяется перед выполнением опера-

[1] Несмотря на наличие слова «алгоритм» в заголовке данного стандарта, символы блок-схем в нем относятся не к алгоритмам, а к схемам данных, программ, работы системы и т.д., наиболее близким к понятию алгоритма можно считать схему работы системы.

[2] Цикл предполагает многократное повторение одной или нескольких (блока) операций – итераций цикла – и часто имеет имя: под именем цикла подразумевается имя ячейки памяти (такая ячейка называется счетчиком цикла), в которой хранится номер итерации цикла (этот номер может увеличиваться или уменьшаться после каждой итерации).

ций цикла (цикл с предусловием), то в верхнем блоке указываются имя цикла и условие завершения, а в нижнем – имя цикла, если же условие завершения проверяется в конце (цикл с постусловием), то в верхнем блоке указывается имя цикла, а в нижнем – условие завершения и имя цикла;

д – символ отображения данных, носитель которых не определен;

е – символ-терминатор, отображающий вход из внешней среды и выход во внешнюю среду (начало и конец работы);

ж – символ-соединитель, отображающий выход в часть схемы и вход в другую часть схемы; символ применяется для разделения схемы на несколько частей; внутри символа указывается обозначение, и соответствующие символы (например, символ, обрывающий схему на странице N, и символ, продолжающий схему на странице N+1) должны иметь одинаковые обозначения;

з – символ для добавления комментариев, комментарии указываются внутри квадратной скобки, а пунктирная линия связывается с комментируемым символом или может обводить несколько комментируемых символов.

Диаграммы Насси-Шнайдермана разработаны в 1972 году аспирантами американского университета Айзеком Насси и Беном Шнайдерманом для структурного программирования. Они получили широкое распространение в Германии (стандарт DIN 66261), по-видимому, вследствие своей наглядности. Основные символы диаграмм Насси-Шнайдермана представлены на рис. 2:

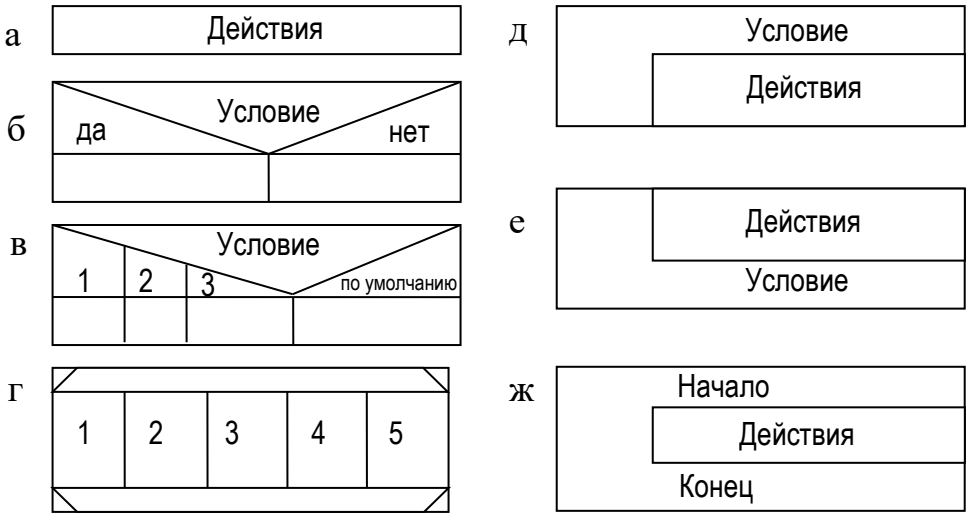


Рис. 2. Символы диаграмм Насси-Шнайдермана

а – символ процесса;

б – символ решения с двумя вариантами;

в – символ решения с несколькими вариантами;

г – символ параллельных процессов, отображает одновременное (параллельное) выполнение нескольких блоков действий;

д – символ цикла с предусловием;

е – символ цикла с постусловием;

ж – символ блока (составного оператора), отображает несколько операторов, помещенных в скобки «начало» и «конец».

Пояснения к символам, соответствующим символам блок-схем, не указаны, так как они даны в расшифровке к рис. 1.

Существуют и другие графические схемы, например, Р-схемы [2], которые можно применять для визуализации алгоритмов.

Псевдокод используется как альтернатива графической схеме алгоритма и представляет собой искусственный язык, обычно являющийся смесью из слов естественного языка, ключевых слов и операций некоторого языка программирования, математических символов, формул и т.п. Псевдокод алгоритма должен быть таким, чтобы его без дальнейшей детализации действий можно было перевести в программный код.

1.2. Об архитектуре ВС

Большинство современных ВС построены в соответствии с архитектурой, представленной Джоном фон Нейманом в 1945 г. в отчете о вычислительной машине EDVAC. Машина фон Неймана состоит из пяти основных компонентов: арифметико-логическое устройство (АЛУ), устройство управления (УУ), память, устройство ввода и устройство вывода. Память состоит из ячеек, каждая из которых имеет адрес или номер, по которому к ячейке можно обратиться, и предназначена для хранения данных и программы. Программа и данные

поступают в память с устройства ввода. АЛУ предназначено для выполнения арифметических и логических операций. УУ поддерживает процесс выполнения команд программы и управляет АЛУ, памятью, устройствами ввода/вывода. АЛУ и УУ вместе составляют центральное устройство, которое в настоящее время называется процессором.

Процессор извлекает из памяти команды в порядке их следования в программе и выполняет их. Необходимые данные также извлекаются из памяти, а результаты, в том числе промежуточные, помещаются в память. Результаты работы программы передаются из памяти в устройство вывода. Особенностью архитектуры фон Неймана является концепция хранимой программы: программа хранится в памяти. При этом память для хранения программ не выделена в отдельный модуль – данные и программный код находятся в одной и той же памяти.

Современная ВС обычно содержит:

– материнскую плату, координирующую работу всех устройств; некоторые устройства могут быть встроены в материнскую плату, например, звуковая карта (микросхема, позволяющая воспроизводить и записывать звук), видеокарта (микросхема, необходимая для формирования изображения на экране монитора);

– один или несколько процессоров;

– энергозависимую оперативную память, в которой хранятся программы во время их выполнения;

– постоянную память, в которой хранятся важные параметры, в том числе информация, необходимая для запуска ВС и загрузки операционной системы[3] в оперативную память;

– вторичную память (память на жестких дисках, флэш-память и другие виды внешней памяти) для долговременного хранения программ и данных;

[3] Операционная система (ОС) – набор программ, выполняющихся на процессоре в течение сеанса работы ВС, управляющих работой всех устройств и предоставляющих пользователю удобный способ взаимодействия с ВС. Распространенные семейства ОС – Windows, UNIX.

– устройства ввода (клавиатура, мышь, сканер и т.д.) и вывода (монитор, принтер и т.д.) для взаимодействия пользователей с ВС.

Некоторыми устройствами ВС управляют контроллеры (специализированные микропроцессоры).

Контроллер может быть встроен в материнскую плату (контроллер клавиатуры) или являться частью устройства (контроллер жесткого диска). Современные ВС имеют шинную архитектуру: устройства через специальные, унифицированные разъемы подключаются к материнской плате и взаимодействуют друг с другом посредством шин – средств передачи и приема информации, физически представляющих собой проводники (группы проводников) определенного вида, например, многожильные кабели. Несколькими шинами управляет контроллер шин.

Процессор является исполнителем ВС, он исполняет инструкции программы, записанной на машинном языке. Инструкция содержит в себе код команды из набора машинных команд процессора, а также может включать один или несколько операндов. Всю совокупность команд, т.е. систему команд процессора можно разбить на следующие группы:

– арифметические команды (сложение, вычитание, умножение, деление);

– команды ввода-вывода, обеспечивающие взаимодействие с внешними устройствами;

– команды переноса данных (перемещение, загрузка);

– логические команды («и», «или», исключающее «или»);

– команды перехода (безусловный переход, вызов подпрограммы).

Операнды являются ячейками памяти или регистрами, содержащими данные, над которыми выполняются операции чтения или записи.

1.3. Средства создания и загрузки программ, компиляция и компоновка

Чтобы перевести алгоритм в программу, которая будет выполняться в ВС, нужно знать язык программирования, на котором создается программа, и иметь в распоряжении инструмент для разработки программного обеспечения. Точнее говоря, два упомянутых средства – язык программирования и инструмент для разработки тесно связаны, так как второе включает реализацию первого.

Язык программирования – это средство взаимодействия программиста с ВС. Он является искусственным языком и обычно включает множество допустимых символов (алфавит), символы и их комбинации для представления операций (например, «+» – сложение), ключевые слова (они используются, в частности, в таких конструкциях, как «решение», «цикл» – см. п. 1.1). Язык программирования определяет правила, по которым в программе создаются структуры для хранения данных, новые операции, команды (операторы), а также другие конструкции, позволяющие записать алгоритм в терминах этого языка.

Первые языки программирования были машинными, программы на таких языках состояли из команд, принадлежащих системе команд применяемой ЭВМ. Т.е. программа на машинном языке непосредственно, без промежуточных преобразований, выполняется исполнителем ВС. Машинные языки называются также низкоуровневыми (или языками первого уровня) и являются машинно-зависимыми из-за того, что ограничены набором команд конкретной машины, иначе говоря, – особенностями ее аппаратной реализации. Позднее появились другие языки, в том числе языки высокого уровня (ЯВУ), не привязанные к аппаратуре ВС. ЯВУ требует либо трансляции написанной на нем всей исходной программы в эквивалентную программу на машинном или близком к машинному языке и последующем выполнении полученной программы, либо покомандного или построчного выполнения исходной программы специально предназначенным для этой цели исполнителем. Второй способ называет-

ся интерпретацией: простой исполнитель-интерпретатор читает каждую команду исходного программного кода на ЯВУ, анализирует и выполняет ее. Транслятор, преобразующий программу на ЯВУ в программу на машинном языке или на языке, близком к машинному, называется компилятором (compiler).

Компиляция в общем случае включает трансляцию содержимого всех модулей программы на ЯВУ в низкоуровневый код в эквивалентных модулях, в частности, в машинный код. Распространены также интерпретаторы компилирующего типа, которые на первом этапе компилируют программу в некоторое промежуточное представление, например, байт-код (машинно-независимый низкоуровневый код). Программа в промежуточном представлении затем интерпретируется интерпретатором, называемым виртуальной машиной.

Язык Си, описываемый в настоящем пособии, относится к компилируемым высокоуровневым языкам. Реализация компилируемого ЯВУ определяется компилятором. Если при компиляции учитываются особенности архитектур различных ВС, то и ЯВУ применяется на различных аппаратных платформах. Можно создать один компилятор, учитывающий особенности различных аппаратных платформ, или несколько различных компиляторов, каждый из которых работает в рамках ВС конкретной архитектуры. Возможности различных компиляторов с данного языка часто унифицируются с помощью соответствующего стандарта. Стандарт задает полное описание – спецификацию языка. Так, например, C17 (ISO/IEC 9899:2018) – последний на текущий момент стандарт языка Си.

Процесс трансляции и загрузки в память программы на языке Си обычно состоит из представленной ниже последовательности основных этапов (детально трансляция описана в п. 4.2) [3].

Вначале исходная программа подготавливается для компиляции программой-препроцессором языка Си. Под процессированием здесь подразумевается компиляция, а препроцессор просматривает исходный код и изменяет его в

соответствии с директивами – специальными, «понятными» только препроцессору командами.

Затем выполняется компиляция, которая переводит исходную программу в программу на языке ассемблера. Это язык низкого уровня, большая часть команд которого является символической записью машинных команд. Поэтому перевод с языка ассемблера в машинные коды выполняется быстро, в отличие от компиляции, представляющей собой сложный процесс разбора исходного кода, в том числе его синтаксический, лексический, семантический анализ. Программа на языке ассемблера транслируется в программу в машинных кодах с помощью транслятора, который, как и сам язык, называется ассемблером, иначе говоря, – программа ассемблируется. При этом создается модуль на машинном языке, называемый объектным файлом (модулем), который обычно содержит неразрешенные ссылки – неопределенные адреса данных и блоков программного кода в других объектных файлах. Если программа состоит из нескольких модулей, то в результате перечисленных действий создается несколько объектных файлов.

На следующем этапе неразрешенные ссылки нужно заменить на реальные адреса, при этом все объектные файлы, используемые программой, кроме файлов динамически подключаемых библиотек (DLL), должны быть объединены и включены в результирующий исполняемый модуль. Таким объединением и настройкой занимается компоновщик (редактор связей, линковщик – linker). Компоновщик представляет собой системную программу, объединяющую независимо оттранслированные с помощью ассемблера программы на машинном языке в исполняемый модуль и настраивающую все неопределенные ссылки [3]. Динамически подключаемые библиотеки содержат программный код, к которому обращается исходная программа, и связываются с программой на этапе ее выполнения. Для загрузки исполняемого файла в память используется специальная программа, загрузчик ОС. Загрузчик выполняет действия по созданию необходимых структур в оперативной памяти ВС для загрузки ис-

полняемого файла и DLL (если они есть), загружает в память исполняемый файл, при наличии DLL – загружает DLL и связывает DLL с программой, после чего передает управление программе.

Компилятор может поставляться отдельно или быть частью специального программного обеспечения, предназначенного для разработки программ. В первом случае, характерном для раннего этапа развития средств вычислительной техники, программный код писался отдельно, в каком-либо текстовом редакторе. Затем файл с программным кодом, имеющий определенное расширение, например, «.c» компилировался, в результате чего создавались объектные модули, имеющие, в частности, расширение «.obj». Они передавались линковщику, который создавал исполняемый файл с расширением «.exe». Такой подход был весьма неудобен: если в процессе компиляции появлялись (синтаксические) ошибки, нужно было снова запускать текстовый редактор, вносить исправления, компилировать, в процессе чего появлялись новые ошибки; невозможно было выполнить программу по шагам и быстро определить смысловые (семантические) ошибки.

Современные инструменты разработки ПО, называемые интегрированными средами разработки (системами программирования), представляют собой графические приложения, содержащие все необходимые модули для создания, загрузки и тестирования программ. В этот перечень входят:

– текстовый редактор и (или) средства создания графических элементов;

– трансляторы (компилятор, ассемблер);

– компоновщик;

– отладчик;

– средства для запуска программы.

Перечисленные средства позволяют создать программный код (с помощью текстового редактора и других вспомогательных средств), скомпилировать и скомпоновать его, запустить программу, а также выполнять, при необходимости, ее отладку, включающую просмотр промежуточных результатов, пошаго-

вое выполнение. Программа может быть консольным или графическим приложением. В консольных программах графические элементы (формы, окна, кнопки и т.д.) не применяются, а взаимодействие пользователя и программы происходит через консоль, эмулирующую текстовый режим: данные программы отображаются в виде символов в текстовом окне.

1.4. Контрольные вопросы и упражнения

Контрольные вопросы

1. Что такое алгоритм?
2. Перечислите основные свойства алгоритма.
3. Перечислите способы описания алгоритма.
4. Изобразите и поясните основные элементы блок-схемы.
5. Изобразите и поясните основные элементы диаграммы Насси-Шнайдермана.
6. Перечислите основные компоненты машины фон Неймана.
7. Из каких устройств состоит современный компьютер?
8. Для чего предназначен процессор?
9. Назовите основные группы машинных команд.
10. Чем отличаются машинные языки программирования от ЯВУ?
11. Что такое интерпретация?
12. Что такое компиляция?
13. Какие действия выполняет компоновщик?
14. Что входит в состав современной интегрированной среды разработки программ?

Упражнения

1. Составьте словесный алгоритм типовых действий обучающегося в университете в течение одного дня.

2. Составьте алгоритм решения уравнения $ax+9x = a^2+6a–18$ относительно x в виде псевдокода.
3. Составьте алгоритм решения уравнения $b^2x – x = b^2 + 5b – 5$ относительно x в виде псевдокода.
4. Составьте алгоритм для решения уравнения $x^2 + 2x - 3 = 0$ в виде псевдокода.
5. В ячейку памяти поместили число 1. Исполнитель 10 раз выполнил следующие действия над этой ячейкой: а) умножить содержимое ячейки на 2, б) записать полученное значение в ту же ячейку, стерев при этом старое значение. Чему равно содержимое ячейки после действий исполнителя?
6. Составьте алгоритм вычисления длины гипотенузы по заданным длинам катетов прямоугольного треугольника в виде блок-схемы.
7. Составьте алгоритм работы продавца весового товара на рынке в течение одного дня в виде диаграммы Насси-Шнайдермана.
8. Составьте словесный алгоритм приготовления борща.
9. Составьте алгоритм вычисления количества отрицательных элементов в последовательности -28, -26, -24, … и представьте его в виде диаграммы Насси-Шнайдермана.
10. Составьте в виде блок-схемы алгоритм определения факта наличия или отсутствия в последовательности 2, 6, 18, … числа а) 54, б) 168, в) 486, г) 576.
11. Последовательность (b_n) треугольных чисел получается из последовательности натуральных чисел так: $b_1 = 1$, $b_2 = 1 + 2$, $b_3 = 1 + 2 + 3$, $b_4 = 1 + 2 + 3 + 4$, … . Составьте алгоритм получения всех треугольных чисел для заданного натурального n в виде псевдокода.
12. Последовательность (c_{2n-1}) квадратных чисел получается из последовательности нечетных чисел 1, 3, 5, 7, … так: $c_1 = 1$, $c_2 = 1 + 3$,

$c_3 = 1 + 3 + 5$, $c_4 = 1 + 3 + 5 + 7$, … . Составьте алгоритм получения всех квадратных чисел для заданного натурального n в виде псевдокода.

13. Робот перемещается по бесконечному полю, разбитому на одинаковые клетки. Исполнитель управляет роботом с помощью команд перехода на одну клетку: вперед, назад, влево, вправо. Приведите словесный алгоритм перемещения робота из исходной клетки вперед по периметру квадрата с длиной стороны n клеток в исходную клетку.
14. Робот перемещается по бесконечному полю, разбитому на одинаковые клетки. Исполнитель управляет роботом с помощью команд перехода на одну клетку: вперед, назад, влево, вправо. Приведите словесный алгоритм перемещения робота из исходной клетки вперед и вверх по диагонали на n клеток.
15. Составьте алгоритм для нахождения суммы первых n элементов последовательности 3, 33, 333, 3333, … в виде блок-схемы.

2. Основы языка Си

2.1. Общие сведения. Линейные программы

2.1.1. Переменные, операторы, операции

Чтобы понять, как устроена программа на языке Си, вспомним о том, для чего она предназначена: с помощью программы исходные (входные) данные должны быть преобразованы в требуемый результат.

Переменные

Данные хранятся в именованных логических ячейках внутренней (оперативной) памяти, называемых переменными. Физически внутренняя память представляет собой линейную последовательность ячеек (байтов), каждая из которых имеет номер или адрес. Обращаться напрямую к байтам из программы на языке высокого уровня не принято и неудобно. Языки программирования высокого уровня, как правило, типизированы: каждый элемент данных должен

иметь свой тип, определяющий множество значений, которые может принимать этот элемент данных, и набор операций над ним. От типа зависит размер элемента данных – он может, например, занимать один байт (символьный тип) или четыре байта (целый тип в тридцатидвухразрядной системе). Так, в тридцатидвухразрядной системе элемент данных знакового целого типа принимает одно из множества значений диапазона от -2^{31} до $2^{31}-1$, над ним определены операции, которые можно выполнять с целым числом – сложение, умножение и т.д. Так как один элемент данных может занимать в памяти группу байтов, целесообразно объединить такую группу в одну логическую ячейку и обращаться к ней по имени. Это и есть переменная. Перед использованием переменную необходимо объявить (декларировать): указать её имя и характеристики, в частности, – имя и тип. Объявление переменной можно совместить с инициализацией – присваиванием начального значения. Над переменной, как над ячейкой памяти, можно выполнить две операции – запись и чтение. Запись значения в переменную называется присваиванием и обозначается в языке Си как «=». Для считывания значения переменной нужно просто указать ее имя:

```
int a = 3, b1 = 2, c2 = 6;
a = b1 + c2;[4]
```
(1)

В примере (1) вначале объявляются переменные a, b1 и c2, объявление совмещено с присваиванием начальных значений (3, 2 и 6 соответственно). Затем переменной a присваивается сумма значений, считанных из переменных b1 и c2: 8 (2 + 6). Предыдущее значение переменной a (т.е. 3) теряется – заменяется на новое в результате присваивания.

Операторы

Подобно тому как продукт какого-либо многостадийного производства проходит несколько этапов обработки, входные данные программы на языке Си изменяются с помощью одного или нескольких последовательных действий над

[4] Здесь и далее в конце примеров не ставятся знаки препинания, относящиеся к тексту данного пособия, так как они могут ввести в заблуждение обучающихся.

ними для получения результата. Такие действия принято называть операторами. Можно провести параллель между оператором языка Си и предложением естественного, например, русского языка. При этом не стоит забывать об ограниченности искусственного языка [языка Си], по сравнению с естественным. Оператор можно рассматривать как лаконичное (короткое, нераспространённое) предложение. Предложение характеризуется завершенностью и заканчивается точкой, вопросительным или восклицательным знаком. Оператор языка Си всегда заканчивается точкой с запятой. Существуют разные типы предложений и, аналогично этому, – разные типы операторов. Предложения по цели высказывания могут быть разных видов, а оператор, как команда языка программирования, – это всегда побудительное предложение. Операторы, как и предложения, можно классифицировать на простые и сложные. Совокупность предложений, связанных по смыслу и грамматически, представляют собой текст. Последовательность операторов, соответствующих алгоритму, образует программу.

Операции

Почти во всех операторах используются операции, например, умножение, извлечение квадратного корня. Операции понимаются в узком смысле[5] как действия над данными, определяемыми программистом. Они делятся на примитивные, встроенные в язык, и на операции, определяемые программистом. При написании программ не всегда возможно ограничиться примитивными операциями, поэтому требуется расширить язык, включив в него новые операции [1]. Они называются подпрограммами, так как состоят из последовательностей операторов и описаний, подобно основной программе. В частности, в языке Си нет встроенных операций ввода и вывода, вместо них применяются заранее написанные подпрограммы ввода и вывода, которые хранятся в библиотеке[6] языка.

[5] Следует отметить, что операция в широком смысле является также действием над данными, определяемыми системой, – это переход, вызов функции, передача параметров и т.п. [1].

[6] В библиотеке языка хранятся заранее подготовленные подпрограммы для определённых часто встречающихся операций, не включенных в язык.

Помимо готовых подпрограмм, хранящихся в библиотеках, в программе можно определить свои собственные подпрограммы и применять их по мере необходимости.

Для того чтобы применить операцию, нужно знать, как она обозначается:

$$b = (a + sqr(1,21))/3; \quad (2)$$

В примере (2) представлен оператор вычисления выражения, в котором использованы операции сложения («+»), деления («/»), присваивания («=»), взятия в скобки («(», «)») и вычисления квадратного корня («sqr»), причем последняя операция реализована в виде подпрограммы. Для обозначения подпрограммы используются имя и информация о входных и выходных данных, необходимых для взаимодействия с другими подпрограммами. В рассмотренном примере число 1,21 – входное данное, число вещественного типа, а выходное данное, как квадратный корень из входного, – число такого же вещественного типа.

В языке Си подпрограмма называется функцией. Основная программа также является функцией и имеет имя «main». Совокупность операторов и описаний, составляющих функцию, называется ее телом и заключается в фигурные скобки. Перед использованием новую функцию необходимо определить. Определение функции в программе – это тело функции, т.е. программный код, плюс заголовок, содержащий её имя, характеристики входных и выходных данных и тип результата, который она выдает. Основная программа, таким образом, является определением функции с именем main и, в частности, выглядит так:

```
int main() {
    … // здесь, внутри фигурных скобок - в теле функции main -
    … // записывается главная программа
    return 0;
}
```

Ключевое слово «int», стоящее слева от имени функции, указывает на то, что функция будет выдавать или возвращать результат целого типа. Это происходит перед ее завершением – функция возвращает 0, что означает успешное завершение: «return 0;». Круглые скобки, записанные справа от имени функции, указывают на то, что имя, после которого они записаны, является именем

функции, а не какого-либо другого объекта. Внутри круглых скобок могут быть перечислены аргументы – входные и выходные данные функции и их характеристики. Даже если аргументов нет, наличие круглых скобок обязательно.

Виды операторов

В [4] перечислены следующие основные виды операторов: описания, вычисления выражения, вызова функции, управления. Во введении были перечислены соответствующие почти всем им группы действий: операторы вычисления выражения соответствуют 1-й группе, управления – 2-4-й группам, вызова функции – 5-й группе. Описания в настоящее время не считаются операторами: операторами называют только те действия, которые связаны с вычислениями и управлением, причем вызовы функций рассматриваются как части выражений. В данном пособии описания иногда причисляются к операторам с целью упрощения изложения, а также ввиду того, что они инициируют определенные действия и в таком контексте являются операторами.

Операторы вычисления выражения показаны в примерах (1) и (2). Выражения в них записаны справа от знака операции присваивания.

Оператор вызова функции представляет собой «просьбу» вызывающей стороны (т.е. «просьбу» функции, из которой происходит вызов) к другой функции выполнить определенную операцию. Подразумевается, что вызывающая функция ждет, пока вызываемая не закончит выполнять операцию. Таким образом, при вызове происходит передача управления в другую функцию, см. (3).

```
#include <stdio.h>
void congratulations()
{
    printf("Congratulations! It's your 1-st step in learning C
    \n");
}
int main()
{
   int a = 5;
   printf("What's this? \n");
   congratulations();
return 0;
}
```
(3)

В программе из примера (3) определена дополнительная функция с именем «congratulations», реализующая операцию вывода на экран символьной строки: «Congratulations! It’s your 1-st step in learning C \n». Вывод, в свою очередь, выполняется с помощью функции стандартной библиотеки языка Си – printf. В основной программе, представленной функцией main, последовательно выполняются четыре оператора: описания и присваивания начального значения переменной a (в одном операторе «int a = 5;»), вызова функции printf для вывода на экран строки «What’s this? \n», вызова функции congratulations и не указанного явно оператора возврата из main. После вызова printf выполнение main приостанавливается, управление передается в тело printf: начинают последовательно выполняться операторы из printf. Последний оператор из printf инициирует возврат в main и возобновление ее выполнения со следующего оператора – вызова функции congratulations. При этом управление вновь передается в вызванную функцию. В данном случае выполняются оператор «printf(″Congratulations! It’s your 1-st step in learning C \n″);» и, не указанный явно, оператор возврата return, возобновляющий работу main. Затем происходит возврат из main – завершение работы программы. Оператор возврата задан явно, «return 0;», так как в заголовке перед main указан тип int – функция main должна возвращать целое число. Определения функции printf нет в программе, оно содержится в библиотечном файле, так как это готовая подпрограмма. Для подключения описания функции к программе используется директива препроцессора «#include <stdio.h>». Функция congratulations определена в самой программе, она имеет заголовок «void congratulations()» и тело, следующее за ним в фигурных скобках. Вызов функции включает ее имя и, как минимум, открывающуюся и закрывающуюся круглые скобки: «congratulations();». По круглым скобкам, которыми заканчивается имя, всегда можно отличить функцию от других объектов.

Операторы управления позволяют разветвлять программы в разных точках на альтернативные ветки операторов, осуществлять многократное повторение операторов и выполнять переходы из одной точки программы в другую.

Описания (декларации)

Декларации (объявления и/или определения) применяются для задания имен, указания характеристик переменных, функций и новых типов.

Объявление переменной задает имя и характеристики переменной. Например, декларация

```
int i;
```

объявляет переменную с именем i целого типа int: она будет хранить целое число. Если декларация инициирует выделение памяти под переменную и присваивание начального значения, то такая декларация называется определением переменной. Так, в (1) представлены определения переменных. В таких случаях говорят, что объявление переменной совмещается с ее определением.

Объявление типа позволяет создать новый тип данных и предполагает присвоение имени какому-либо существующему или создаваемому составному типу. Например, можно назвать тип int новым именем – integer (т.е. создать integer в качестве синонима int), применяя ключевое слово typedef:

```
typedef int integer;
```

Объявление функции состоит из ее заголовка, а определение – это тело плюс заголовок, как было сказано выше:

```
//объявление функции – только заголовок
void my_func(int a);
//определение функции – заголовок и тело в фигурных скобках:
void my_func(int a) { printf("a=%d",a); }
```

Объявление может содержать один или несколько деклараторов. Декларатор – элемент объявления, который определяет имя объекта или функции. Например, в объявлении

```
int *p, a;
```

есть 2 декларатора: *p и a. В объявлении

```
double **a[n][m];
```

элемент **a[n][m] – декларатор. Декларатор в объявлении одного объекта – это идентификатор плюс все символы справа до точки с запятой плюс все символы слева до пробела, т.е. декларатор – имя плюс относящиеся к нему символы справа и слева.

2.1.2. Структура программы. Введение в ввод/вывод. Линейные программы

Структура программы

Таким образом, основная последовательность операторов, которая начинает выполняться после запуска программы на языке Си, содержится в функции main. Помимо функции main программа содержит описания данных и операций, в том числе определения самостоятельно подготовленных функций, а также специальных указаний – директив препроцессора, предназначенных для обработки программного кода до компиляции.

Пример:

```
#include <stdio.h>      /* директивы препроцессора include */
#include <math.h>
/* определение собственной функции */
double mydegree(double a, double b) {
     double res = pow(a, b);
     return res;
}
int main()  {      /* основная функция */
    double x, y, z;/* оператор описания переменных x, y, z */
    printf("Input the value and degree: \n");
    printf("value = "); /*операторы вызова функций: */
    scanf("%lf", &x);   /* printf, printf, scanf, */
    printf("degree = "); /* printf, scanf */
    scanf("%lf", &y);
    z = mydegree(x, y) + 2;/*оператор вычисления выражения,
                             в котором есть вызов функции*/
    /*вызов printf для вывода результата */
    printf("%.3lf to the %.1lfth is %.3lf", x, y, z);
    return 0;
}                                                          (4)
```

Программа из примера (4) содержит две директивы include, собственную операцию возведения числа в степень, представленную определением функции mydegree, и основную часть программы, представленную девятью операторами в функции main. В программе есть предложения, которые не обрабатываются компилятором, а пропускаются, – это комментарии. Каждый комментарий в (4) начинается комбинацией символов «/*» и заканчивается комбинацией «*/». Такие комментарии задаются в соответствии с ANSI C, первым стандартом для языка Си. Начиная со стандарта C99 можно применять однострочные комментарии, начинающиеся комбинацией «//». Директивы препроцессора include позволяют включить в программу тексты заголовочных файлов с расширением «.h» с целью вызова сторонних функций printf, scanf и pow из библиотек готовых подпрограмм (printf и scanf относятся к stdio.h, а pow – к math.h[7]).

Трансляция программы на языке Си может выполняться по частям, если текст программы хранится в нескольких исходных файлах. Исходный файл вместе с заголовками и другими исходными файлами, включенными в него с помощью директивы #include, называется модулем, подлежащим препроцессированию, так как каждый исходный файл должен быть обработан препроцессором. После обработки препроцессором каждый такой файл называется модулем, подлежащим трансляции. Оттранслированные модули могут храниться отдельно или в библиотеках. Отдельные модули, подлежащие трансляции, взаимодействуют друг с другом посредством обращения к функциям (в частности, с помощью обращений из одного модуля к функциям другого) и манипуляций с объектами или файлами данных. Модули могут транслироваться отдельно и затем вместе связываться в выполняемую программу. Порядок обработки исходного файла при трансляции читатель может найти в данном пособии после описания элементов синтаксиса.

[7] В стандартной библиотеке языка Си в заголовочных файлах (с расширением «.h») указаны имена и характеристики функций, определения функций содержатся в библиотечных файлах.

Введение в ввод/вывод

Прежде чем начать обработку данных, необходимо задать их начальные значения. Это осуществляется посредством ввода данных с клавиатуры. Результаты работы программы и какие-либо промежуточные результаты обычно выводят на экран. В языке Си нет встроенных операций для ввода и вывода. Вместо этого для ввода применяется функция scanf (scanf_s), для вывода – функция printf стандартной библиотеки языка Си. Для подключения библиотеки нужно добавить в начало программы директиву

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

Функциям ввода/вывода из вызывающей функции передаются входные данные, которые называются аргументами или параметрами. Обе функции в качестве первого аргумента принимают так называемую строку формата, записанную в кавычках, например:

```
printf("Это строка формата со спецификацией %d\n", some_variable);
```

Строка формата может содержать обычные символы, а также специальные комбинации символов, которые позволяют управлять вводом/выводом.

С символа "%" в вызовах функций printf и scanf обычно начинается так называемая спецификация формата или указатель формата. Спецификация формата указывает на то, каким образом должны быть проинтерпретированы введенные с клавиатуры или выводимые на экран данные: как целое, или как вещественное, или как код символа, или как строка, в частности: %c – одиночный символ; %s – строка символов; %d,%i – целое со знаком, %u – беззнаковое целое, %f,%F,%e,%E,%g,%G,%a,%A – вещественное, %lf, %le, %lg – двойное вещественное (double), %Lf, %Le, %Lg – длинное двойное вещественное (long double). Чтобы ввести или вывести число в вещественном формате с заданным количеством знаков после запятой, нужно писать вместо %f – %.rf (%.rlf для double), где r – количество знаков. Например, %.2f – с двумя знаками после запятой, %.5f – с пятью знаками после запятой.

С помощью функции scanf можно вводить значения одной или нескольких переменных с клавиатуры:

```
scanf("%d", &a);    //пользователь введет число,
                    //и оно запишется в переменную a
scanf("%f%f", &b, &c);//пользователь введет последовательно
                      //2 числа, первое запишется в b, второе – в c
```

В первом примере в функцию scanf передаются через запятую два аргумента: "%d" (строка формата) и &a. Аргумент "%d" является указанием на то, что введенное с клавиатуры число будет проинтерпретировано как целое (из-за буквы d после %). Второй аргумент – адрес переменной, по которому запишется введенное значение. Во втором примере первый аргумент "%f%f" указывает на то, что и первое (первое сочетание %f), и второе (второе сочетание %f) числа будут проинтерпретированы как вещественные. Так как спецификаций формата две, то и адресов для записи данных должно быть два: &b, &c.

Функция printf выводит на экран строку символов, в которую преобразуется строка формата, передаваемая в printf в качестве первого аргумента. Строка формата также может содержать спецификации формата и другие специальные последовательности символов, которые не выводятся, а используются для управления выводом. Если строка формата не содержит никакие специальные символы, она выводится как есть. Например, вызов printf("We like programming"); напечатает на экране строку «We like programming» (без кавычек, кавычки в printf – элемент синтаксиса). Если строка содержит так называемые escape-последовательности символов (см. п. 2.2.1), осуществляются некоторые действия: например, комбинация \n в строке формата вызовет перевод курсора в начало следующей строки. Комбинация \t (горизонтальная табуляция) вызывает перевод курсора из текущей позиции на несколько символов вперед (пропуск группы пробелов). Например, вызов printf("Horizontal\ttab\n"); выведет на экран строку «Horizontal», переведет курсор на несколько позиций вперед, после чего выведет «tab» и переведет курсор в начало следующей строки, так что текущей позицией окажется первый символ следующей строки на экране:

```
Horizontal      tab
_
```

Вызов printf("%d", a) напечатает на экране значение переменной a в формате d – как целое. Спецификация %d указывает на то, что в текущей позиции вместо %d нужно напечатать значение переменной a в формате «целое» (d). Пусть значения переменных a, b, c, e равны соответственно 1, 2, 3, 49. Если требуется вывести одновременно символы и значения переменных, то обычные символы, спецификации формата и escape-последовательности вида \t комбинируются:

```
printf("a = %d\tb = %d\n c = %f, e = %c.", a, b, c, e);
```

В результате указанного выше вызова printf строка формата проинтерпретируется следующим образом: «a = » – просто символы, выводятся без изменений, вместо %d напечатается значение переменной a (второй аргумент printf), вместо \t подставится несколько пробелов, «b = » – просто символы, вместо %d – значение b, вместо \n – перевод курсора в начало следующей строки, «c = » – без изменений, вместо %f – значение c в вещественном формате, «, e = » – без изменений, вместо %c – значение e в виде символа (49 – код символа «1», поэтому выведется 1), точка – без изменений:

```
a = 1         b = 2
c = 3.000000, e = 1.
```

Создание линейных программ на языке Си

Здесь рассматривается задача реализации вычислений по заданной формуле. Используются простые операции: * (умножение), / (деление), + (сложение), - (вычитание), = (присваивание). Для выполнения операций извлечения корня, возведения в степень, вычисления косинуса (и т.п. – в случае других формул) понадобится библиотека готовых математических функций, с этой целью в программу нужно включить заголовочный файл math.h. Вызов математической функции содержит ее имя и в круглых скобках аргумент(-ы) – величина(-ы), над которой(-ыми) нужно выполнить соответствующую операцию. Если

аргументов несколько, они записываются через запятую. Например, pow(7.0, 5) – возведение числа 7.0 в степень 5. Функция выдаёт (возвращает) результат, который можно записать в какую-либо переменную:

```
double r = pow(1.8, 4); //вычислить 1,8⁴; результат записать в r.
```

Наиболее распространённые функции:

- abs(x) – вычисляет модуль аргумента x (типа int);
- fabs(x) – вычисляет модуль x (типа double);
- exp(x) – вычисляет e^x (x типа double);
- log(x) – вычисляет натуральный логарифм x (типа double);
- log10(x) – вычисляет десятичный логарифм x (типа double);
- cos(x) – вычисляет косинус x (типа double), заданного в радианах;
- sin(x) – вычисляет синус x (типа double), заданного в радианах;
- tan(x) – вычисляет тангенс x (типа double), заданного в радианах;
- asin(x) – вычисляет арксинус x (типа double в диапазоне от -1 до 1);
- acos(x) – вычисляет арккосинус x (типа double в диапазоне до -1 до 1);
- atan(x) – вычисляет арктангенс x (типа double);
- sqrt(x) – вычисляет квадратный корень из x (типа double);
- pow(x, y) – возводит x (типа double) в степень y (типа double);
- fmod(x, y) – вычисляет остаток от деления x (типа double) на y (типа double), возвращает результат типа double;
- floor(x) – находит наибольшее целое, не превышающее x (типа double), возвращает результат в форме double.

Ниже представлена простая линейная программа для выполнения вычислений по формуле: $y = 11x^{19} - 13\sqrt{x} + 7\cos(3x)$ с вводом x и выводом y.

```
#include <stdio.h>
#include <math.h>
int main() {
   double x, y;        //объявление переменных x, y
   printf("x = ");     //вывод на экран подсказки: x =
   scanf("%lf", &x);   //ввод x в формате double (%lf)
   y = 11*pow(x, 19.) - 13*sqrt(x) + 7 * cos(3*x);//расчет y
```

```
    printf("y = %lf", y); //вывод результата в формате double
    return 0;             //нормальное завершение main
}
```

2.1.3. Контрольные вопросы и упражнения

Контрольные вопросы

1. Что такое переменная?
2. Что представляет собой внутренняя память физически?
3. Какие ограничения накладывает тип на данные?
4. Назовите основные операции, которые можно выполнить над переменной.
5. Что представляет собой оператор языка Си?
6. Что представляет собой операция?
7. Как называют операции, которые не встроены в язык Си?
8. Для чего предназначена подпрограмма и как она называется в языке Си?
9. Из чего состоит определение функции в программе на языке Си?
10. Где указывается тип значения, которое возвращает функция, и что означает фраза «функция возвращает значение»?
11. Что такое декларация (описание или оператор описания)?
12. Что задает объявление переменной?
13. Что задает объявление типа?
14. Что включает объявление функции?
15. Что представляет собой оператор вызова функции?
16. Что позволяют делать операторы управления?
17. Назовите основные элементы программы на языке Си?
18. С какой целью в программу на языке включают заголовочные файлы?
19. Каков общий порядок трансляции программы на языке Си?

20. Для чего применяется функция scanf (scanf_s), что представляет собой ее первый аргумент?
21. Для чего применяется функция printf, сколько у неё может быть аргументов?
22. Что такое спецификация формата, назовите спецификации формата для данных разных типов.
23. Приведите примеры escape-последовательностей символов.
24. Перечислите наиболее распространенные математические функции, доступные после включения в программу файла math.h.

Упражнения

1. Составьте программу, в которой вводятся значения переменных a, b, c типа int, переменных d, e типа double. Вычисляется сумма s = a+b+c, разность r = d-e и выводятся на экран в виде:
   <значение_a> + <значение_b> + <значение_c> = <значение_s>;
   <значение_d> - < значение_e> = <значение_r>.
   Здесь <значение_a> – число, которое записано в переменной a, <значение_b> – число, записанное в переменной b, и т.д., т.е. вывод может быть таким:
   3 + 41 + 5 = 49;
   4.56 – 3.32 = 1.24.
2. Напишите программу для решения уравнения $ax+9x = a^2+6a-18$ относительно x. Значение a вводится пользователем, x – выводится.
3. Напишите программу для решения уравнения $b^2x - x = b^2 + 5b - 5$ относительно x. Значение b вводится пользователем, x – выводится.
4. Напишите программу для вычисления y по формуле: $y = 8x^{10} - 6x^6 + 5x^5 - \sqrt[4]{(x^2 - 1)}$, x вводится пользователем, y выводится.

5. Напишите программу для вычисления y по формуле: $y = 10x^9 + 5x^8 - 3x^2 + \sqrt[3]{(x^2+1)}$, x вводится пользователем, y выводится.
6. Напишите программу для вычисления y по формуле: $y = 15x^8 - 2x^7 - x + \sqrt[5]{x^3}$, x вводится пользователем, y выводится.
7. Напишите программу для вычисления y по формуле: $y = 3x^{10} - 4x^9 + 5x^8 - \sqrt[7]{(x^5+4x)}$, x вводится пользователем, y выводится.
8. Напишите программу для вычисления y по формуле: $y = 2x^7 - 3x^6 + 4x^5 - 1/\left(x^3 - 2\right)$, x вводится пользователем, y выводится.
9. Напишите программу для вычисления y по формуле: $y = 28x^{10} - 13x^5 + 4x^2 - \sqrt{x}/2$, x вводится пользователем, y выводится.
10. Напишите программу для вычисления y по формуле: $y = 16x^9 + 15x^8 + 9x^5 - x + \sqrt[13]{x^5}$, x вводится пользователем, y выводится.
11. Напишите программу для вычисления y по формуле: $y = 2x^{10} - 3x^5 - 2x - \cos(2x)$, x вводится пользователем, y выводится.
12. Напишите программу для вычисления y по формуле: $y = -22x^{12} - 12x^6 - 6x^3 + \sqrt[16]{x^3}$, x вводится пользователем, y выводится.
13. Напишите программу для вычисления y по формуле: $y = 15x^9 - 6\sin(2x) + 5\cos(x)$, x вводится пользователем, y выводится.

14. Напишите программу для вычисления y по формуле: $y = 3x^{11} - 2x^7 + x^5 - \sqrt[5]{(x^3 + 3x)}$, x вводится пользователем, y выводится.
15. Напишите программу для вычисления y по формуле: $y = 2x^{10} - 3x^5 - 2x \ - \cos(2x)$, x вводится пользователем, y выводится.

2.2. Лексические элементы

2.2.1. Основные элементы алфавита[8]

Базовое множество символов

В языке Си используется два множества символов: символы, допустимые в исходном коде – исходный набор символов, и символы, допустимые в среде, в которой исполняется программа – исполняемый набор символов.

Каждое множество далее делится на базовое множество символов и множество расширенных символов. Базовые символы исходного и исполняемого набора составляют подмножество из 96 однобайтовых символов, к которым относятся:

а) 26 заглавных и 26 прописных букв латинского алфавита (A-Z, a-z);

б) 10 десятичных цифр (0-9);

в) представленные ниже 29 графических символов:

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

г) пробел и escape-последовательности символов, представляющие вертикальную табуляцию, горизонтальную табуляцию, символ перевода страницы и символ перевода строки.

[8] При описании элементов языка Си использовалась последняя версия черновика стандарта C11 от 12 апреля 2011 года [http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf] и [5].

Одна и та же строчная и прописная буквы являются различными для компилятора. В файлах исходного кода применяется некоторый индикатор перехода на новую строку, в частности, одиночный символ конца строки. В базовом исполняемом наборе символов должны быть escape-последовательности символов: «\a», «\b», «\r» и «\n», описание которых приведено ниже.

Escape-*последовательности символов*

Управляющие последовательности символов или escape-последовательности начинаются с символа обратный слэш (\ ). Большая часть из них предназначена для обозначения неграфических символов, которые вызывают определенные действия:

а) \a – «сигнал» – производит звуковой или визуальный сигнал;

б) \b – «забой» – перемещает курсор (курсор – мигающий символ на экране, обозначающий текущую позицию, обычно это знак нижнего подчеркивания) в предыдущую позицию, стирая текущий символ;

в) \f – «новая страница» – перемещает курсор в начало следующей логической страницы;

г) \n – «новая строка» – перемещает курсор в начало следующей строки;

д) \r – «возврат каретки» – перемещает курсор в начало текущей строки;

е) \t – «горизонтальная табуляция» – перемещает курсор вперед, пропуская несколько (группу) позиций;

ж) \v – «вертикальная табуляция» – перемещает курсор вниз, пропуская группу позиций.

В отдельных случаях обратный слэш ставится перед символом, чтобы символ интерпретировался (печатался) как есть: \', \", \?, \\, а не как элемент синтаксиса. Например, для вывода на экран строки «Привет, мир!» в кавычках нужно выполнить оператор «printf("\"Привет, мир!\"");», в котором внешние кавычки в круглых скобках являются элементами синтаксиса, а вместо каждой последовательности \" на экран выводится кавычка (без слэша). Еще несколько escape-последовательностей используется для представления восьмеричных и

шестнадцатеричных констант, универсальных символов (см. п. 2.2.2). Символы пробел, табуляция, перевод строки, возврат каретки, новая страница, вертикальная табуляция и новая строка называются пробельными, так как они имеют то же назначение, что и пробелы между словами и строками в тексте на естественном языке.

2.2.2. Данные простых типов

В программах на языке Си данные представляются переменными или константами. Значения констант остаются постоянными, а переменных – изменяются во время работы программы.

Базовые типы

Объект, функция и выражение имеют свойство, называемое типом, который определяет то, каким образом интерпретируется двоичное значение, хранящееся в объекте или являющееся результатом вычисления выражения. Каждый элемент данных может быть элементом простого (базового) типа данных или составного типа, создаваемого на основе простых типов. К базовым типам, применяемым для создания элементов данных, относятся следующие группы типов данных (там, где указано, в скобках представлены минимальный размер переменной в битах и соответствующий диапазон изменения):

– 5 стандартных целых типов со знаком: signed char (знаковый символьный, 8 бит, от -127 по 127), short int (короткий целый, 16 бит, от $-(2^{15}-1)$ по $2^{15}-1$), int (целый, 16 бит, от $-(2^{15}-1)$ по $2^{15}-1$), long int (длинный целый, 32 бита, от $-(2^{31}-1)$ по $2^{31}-1$), long long int (двойной длинный целый, 64 бита, от $-(2^{63}-1)$ по $2^{63}-1$), а также могут быть зависящие от реализации расширенные (extended) целые типы со знаком;

– стандартные беззнаковые (unsigned) целые типы, включающие тип _Bool (логический) и беззнаковые типы, соответствующие целым типам со знаком: unsigned char (8 бит, от 0 по 255), unsigned short int (16 бит, от 0 по $2^{16}-1$),

unsigned int (16 бит, от 0 по 2^{16}-1), unsigned long int (32 бита, от 0 по 2^{32}-1), unsigned long long int (64 бита, от 0 по 2^{64}-1), а также могут быть зависящие от реализации расширенные беззнаковые целые типы;

– три типа для действительных чисел с плавающей точкой: float (вещественный), double (двойной вещественный), long double (длинный двойной вещественный);

– три типа для комплексных чисел (их наличие или отсутствие зависит от реализации компилятора): float _Complex (комплексный), double _Complex (двойной комплексный), long double _Complex (длинный двойной комплексный).

Для целых типов стандарт языка Си гарантирует, что

1 = sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long).

Описание переменных

Каждая переменная перед использованием должна быть объявлена. Объявление переменной включает ее идентификатор (имя) и тип. Идентификатор – это последовательность, состоящая из нецифровых символов (строчных и заглавных букв латинского алфавита, считающихся различными, знака «_», других символов, определенных реализацией) и цифр (0-9), начинающаяся не с цифры. Идентификатор не должен совпадать с каким-либо из ключевых слов или именем функции из подключаемой библиотеки.

Пример:

```
int val_int;
```

задает описание переменной с именем val_int, в которой могут храниться целые числа со знаком и ноль.

Описание констант. Целые константы

Каждая константа может иметь тип, при этом ее значение должно принадлежать диапазону значений, задаваемому данным типом. Целые константы бывают десятичными, восьмеричными и шестнадцатеричными. Каждая кон-

станта может иметь суффикс (окончание), указывающий на ее тип. В табл. 1 представлены списки типов констант, возможные суффиксы и их комбинации.

Таблица 1

Суффиксы целых констант и списки их типов

Суффикс	Списки типов	
	Десятичная константа	Восьмеричная или шестнадцатеричная константа
нет	int, long int, long long int	int, unsigned int, long int, unsigned long int, long long int, unsigned long long int
u или U	unsigned int, unsigned long int, unsigned long long int	unsigned int, unsigned long int, unsigned long long int
l или L	long int, long long int	long int, unsigned long int, long long int, unsigned long long int
(u или U) и (l или L)	unsigned long int, unsigned long long int	unsigned long int, unsigned long long int
Ll или LL	long long int	long long int, unsigned long long int
(u или U) и (ll или LL)	unsigned long long int	unsigned long long int

Константа получает первый по списку подходящий тип. Если для константы нет подходящего типа из списка, возможно, ей будет соответствовать расширенный целый тип. Если же константа не имеет и расширенного типа, то она считается не имеющей типа.

Десятичная константа начинается с ненулевой цифры и может содержать десятичные цифры, например: 14, 20, 126.

Восьмеричная константа начинается с цифры 0, после которой записываются восьмеричные цифры (0, 1, 2, 3, 4, 5, 6, 7), например: 0164, 077.

Шестнадцатеричная константа имеет префикс «0x» или «0X» и может содержать шестнадцатеричные цифры, к которым относятся 10 десятичных цифр

и первые шесть строчных (a, b, c, d, e, f) или заглавных (A, B, C, D, E, F) букв латинского алфавита, например: 0xf4, 0XA1b.

Константы с плавающей точкой

Константы с плавающей точкой бывают десятичные и шестнадцатеричные. Общей (но не обязательной) формой представления констант с плавающей точкой является следующая:

$$Aq^{\pm p},$$

где A – рациональное число, называемое значащей частью или мантиссой, q – используемое основание (10 – для десятичных, 2 – для шестнадцатеричных констант), p – порядок, показатель степени, в которую возводится q, $q^{\pm p}$ называется экспоненциальной частью или экспонентой. В частности, рациональное число 24,345 может быть представлено как $0{,}024345 \cdot 10^3$ или $2434{,}5 \cdot 10^{-2}$. Порядок для таких констант в тексте программы указывается с помощью символа порядка, «e» («E») или «p» («P»), и его целочисленной величины, возможно, отрицательной.

В целом, константа с плавающей точкой имеет значащую часть, за которой может следовать экспоненциальная часть и суффикс, указывающий на тип константы. Компоненты значащей части могут включать последовательность цифр целой части, период - «.», а также последовательность цифр дробной части. Экспоненциальная часть включает символ экспоненты: «e», «E», «p» или «P», за которым следует экспонента, состоящая из последовательности цифр, возможно, со знаком. Как минимум, целая или дробная часть должна присутствовать. Для десятичных констант должен присутствовать период или экспоненциальная часть.

Формат десятичной константы с плавающей точкой (в квадратных скобках здесь и далее показаны необязательные элементы, в угловых скобках указано, что должно содержаться в соответствующих позициях):

<цифры>.[< e или E >[< + или – >]<порядок>][<суффикс>], или

[<цифры>].<цифры>[< e или E >[< + или – >]<порядок>][<суффикс>], или
<цифры>< e или E >[< + или – >]<порядок>[<суффикс>],
где цифры – десятичные цифры представляемого действительного числа; порядок – десятичные цифры порядка, порядок представляет собой степень числа 10; суффикс – одна из букв «f», «l», «F», «L».

Формат шестнадцатеричной константы с плавающей точкой:
<0x или 0X ><цифры>.< p или P >[< + или – >]<порядок>[<суффикс>], или
<0x или 0X >[<цифры>].<цифры>< p или P >[< + или – >]<порядок>[<суффикс>], или
<0x или 0X><цифры>< p или P >[< + или – >]<порядок>[<суффикс>], где
цифры – шестнадцатеричные цифры представляемого действительного числа; порядок – десятичные цифры порядка, порядок представляет собой степень числа 2; суффикс – одна из букв «f», «l», «F», «L». Константы с плавающей точкой без суффикса имеют тип double. Если суффикс задан буквой f или F, константа имеет тип float, если же – l или L – тип long double.

Примеры констант с плавающей точкой:

- 12.24e2 (1224,0) – десятичная константа типа double;
- -15.444E-4f (-0,0015444) – десятичная константа типа float;
- 0x1.52A1Fp0L($1{,}52A1F_{16}$) – шестнадцатеричная константа типа long double.
- 0x2.5ep+1($4{,}BC_{16}$) – шестнадцатеричная константа типа double.

Перечисляемые константы

Перечисляемая константа представляет собой идентификатор, имеющий тип int.

Символьные константы

Символьная константа – заключенная в одиночные кавычки (апострофы) последовательность из одного или более символов или управляющий символ (escape-последовательность) в апострофах. Если константа содержит один символ, ее величина равна значению соответствующего кода символа: символы в

памяти ВС представляются в виде чисел или кодов – каждому символу соответствует уникальный код.

Соответствия кодов символам реализуются с помощью стандартов кодирования (ASCII, Unicode), которые задают алгоритмы преобразования символа в его код или просто содержат таблицы однозначного соответствия кодов (целых неотрицательных чисел) символам. Первые простые кодировки (ASCII, EBCDIC) как раз представляли собой такие таблицы. Одним из наиболее распространенных стандартов, для которого многие современные кодировки сохраняют совместимость, является ASCII (American Standard Code for Information Interchange), ставящий в соответствие основным символам семибитный код. Стандарт Unicode посредством таблиц множества универсальных символов (Universal Coded Character Set, UCS) позволяет использовать двухбайтные (UCS-2) и четырехбайтные (UCS-4) коды универсальных символов фиксированной длины. Unicode также содержит кодировки (UTF – Unicode transformation format), позволяющие использовать многобайтные коды символов переменной длины: разным символам могут быть сопоставлены коды разной длины. В частности, в кодировке UTF-8 символы, коды которых меньше 128, имеют однобайтный код, соответствующий ASCII-символам, остальные символы имеют длину от двух до четырех байт; в UTF-16 символ занимает 2 или 4 байта, в UTF-32 – 4.

Стандартная библиотека (начиная с C11) определяет с помощью typedef синонимы wchar_t, char16_t и char32_t для представления расширенных символов:

- wchar_t – определяемый реализацией (конкретным компилятором) тип данных стандарта ANSI/ISO C для расширенных символов. В компиляторе Microsoft представляет 16-битный расширенный символ кодировки UTF-16LE (используемой в Windows);
- char16_t – целочисленный тип без знака, используемый для 16-битных расширенных символов UTF-16, синоним uint_least16_t (из stdint.h);

– char32_t – целочисленный тип без знака, используемый для 32-разрядных расширенных символов UTF-32, синоним uint_least32_t.

Исходный для языка Си символьный тип, тип char, зависит от реализации и может представлять ASCII-символ, или символ из любого набора ISO-8859, или отдельный байт многобайтового символа, или символ в кодировке UTF-8 Unicode. Обычно тип char – целочисленный 8-битный тип.

Формат символьных констант целого и расширенных типов:

- *целого типа*: 'c-последовательность' (тип int);

- *расширенных типов*:

- L 'c-последовательность' (тип wchar_t),
- u 'c-последовательность' (тип char16_t),
- U 'c-последовательность' (тип char32_t),

где 'c-последовательность' состоит из одного или нескольких 'c-символов', а 'c-символ' – любой символ из исходного набора, кроме апострофа ('), обратного слэша (\ ) и символа новой строки, или escape-последовательность.

Формат escape-последовательностей:

1. Простые: одна из \', \", \?, \\, \a, \b, \f, \n, \r, \t, \v.

2. Восьмеричные: \ одна, две или три восьмеричные цифры.

3. Шестнадцатеричные: \x шестнадцатеричные (16-е) цифры.

4. Коды универсальных символов Unicode:

\uhhhh,

\Uhhhhhhhh,

где h – шестнадцатеричная цифра (в том числе A-F, a-f).

Коды универсальных символов могут использоваться в идентификаторах, символьных константах и строковых литералах для обозначения символов, которых нет в базовом множестве символов. Короткий код такого символа не может быть меньше, чем 00A0, кроме 0024 ($), 0040 (@) и 0060 (`), а также входить в интервал от D800 до DFFF включительно. Комбинация \Uhhhhhhhh содержит восемь 16-х цифр, \uhhhh – четыре 16-е цифры hhhh (или восемь 16-х

цифр 0000hhhh). В зависимости от того, какой тип имеет соответствующая переменная или константа, код универсального символа может отображаться («превращаться») в более чем один символ. Например, код \U0001f34c в UTF-8 отображается в 4 кодовые единицы типа char: \xF0\x9F\x8D\x8C, а в UTF-16 – в 2 кодовые единицы \uD83C\uDF4C [5], правило отображения универсального символа, в частности, в UTF-8 см. в стандарте (документе) RFC 3629.

Двойные кавычки (") и знак вопроса (?) могут также быть представлены без предшествующего слэша (как обычные символы), но апостроф представляется только как escape-последовательность (\'). Символы, не входящие в исходное множество символов, представляются с помощью кодов универсальных символов. Восьмеричные и шестнадцатеричные константы без префикса имеют тип unsigned char, с префиксом L – беззнаковый тип, соответствующий wchar_t, с префиксом u – char16_t, с префиксом U – char32_t. Для представления нуль-символа, применяемого в качестве признака конца строки и в других случаях, используется комбинация \0.

Строковые литералы

Литерал – элемент программы, непосредственно представляющий значение. Описанные выше константы также называют литералами: например, целочисленные литералы, литералы с плавающей точкой.

Строковый литерал (константная строка) имеет следующий формат:
[<префикс кодировки>] [<последовательность s-символов>]

Префикс кодировки – один из элементов: «u8», «u», «U», «L»; s-символом является любой символ из исходного набора символов, за исключением символов двойной кавычки ("), обратного слэша (\) и символа новой строки (\n), или escape-последовательность (в том числе универсальный символ).

Символьный строковый литерал – последовательность из нуля или более символов типа char, заключенных в двойные кавычки, например, «asdf». Строковый литерал UTF-8 – то же, с префиксом u8. Расширенный строковый литерал – то же, с префиксом L, u или U. В случае префикса L символы в строковом

литерале имеют тип wchar_t, префикса u – тип char16_t, префикса U – тип char32_t. Последовательность соседних строковых литералов не может одновременно включать расширенные строковые литералы и литералы в кодировке UTF-8. В строковых литералах, в отличие от символьных констант, апостроф может быть представлен и как символ: "в конце этой строки есть апостроф' ", и как escape-последовательность: "в конце этой строки есть апостроф\' ", а двойная кавычка только как escape-последовательность: "примерно в середине \"этой строки есть кавычки".

Во время трансляции смежные строковые литералы, т.е. литералы, разделенные только пробелами, объединяются посредством конкатенации (сцепления). Объединение выполняется для пар расширенных литералов или для пар «узких» литералов. Если один литерал не имеет префикса, результирующий строковый литерал имеет ширину/кодировку, определяемую литералом с префиксом. Если два строковых литерала имеют разные префиксы, конкатенация определяется реализацией. Каждая строка в программе на языке Си должна иметь последним символом нуль-символ (\0). Поэтому на следующем этапе трансляции в конец каждого готового строкового литерала добавляется нуль-символ, и затем литерал сохраняется в статической памяти.

2.2.3. Ключевые слова, знаки пунктуации, имена заголовочных файлов и комментарии

Ключевые слова

Ключевые или зарезервированные слова – идентификаторы, которые предназначены для задания типов переменных, реализации конструкций языка Си, их нельзя использовать в других целях. Список ключевых слов:

```
auto        continue    enum
break       default     extern
case        do          float
char        double      for
const       else        goto
```

```
if                  static              _Alignof
inline              struct              _Atomic
int                 switch              _Bool
long                typedef             _Complex
register            union               _Generic
restrict            unsigned            _Imaginary
return              void                _Noreturn
short               volatile            _Static_assert
signed              while               _Thread_local
sizeof              _Alignas
```

Знаки пунктуации

Знак пунктуации – один из представленных ниже символов или их комбинаций:

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

В зависимости от контекста, знак пунктуации может вызывать выполнение той или иной операции. Например, знак минус может использоваться как унарный минус: «c = -b+a;» или как бинарная операция вычитания: «c = a - b;».

Некоторые знаки пунктуации недоступны, если программа использует множество символов одного из стандартов ISO 646:1983. В таких случаях для указания того или иного знака пунктуации используется определенная группа символов. В частности, комбинации из пар символов <: :> <% %> %: %:%: называют диграфами; они приводят к тем же действиям, что и знаки [ ] { } # ## соответственно. Т.е. перечисленные диграфы – альтернативы указанным знакам пунктуации. Существуют также трёхсимвольные группы – триграфы (табл. 2).

Таблица 2

Соответствие знаков пунктуации триграфам

Знак пунктуации	{	}	[	]	#	\	^	|	~
Триграф	??<	??>	??(	??)	??=	??/	?? '	??!	??-

Если разработчик программы использует триграфы, то на первом этапе трансляции, до распознавания строковых литералов и комментариев, они заменяются на соответствующие одиночные символы.

Имена заголовочных файлов

Имена заголовочных файлов являются частью директив препроцессора и имеют следующий формат:

<один или несколько h-символов> или

"один или несколько q-символов",

где h-символ – любой символ из исходного набора, кроме \n и >, q-символ – любой символ из исходного набора, кроме \n и ".

Комментарии

Комментарии нужны лишь как пояснения для людей, читающих программный код, компилятором они игнорируются. Существует 2 типа комментариев:

1) /* здесь, возможно на нескольких строках, помещается сам комментарий */

2) // однострочный комментарий

Первый тип – многострочный комментарий в стиле Си – комментируемый текст окаймляется комбинациями /* и */. Последовательность /* начинает комментарий, если она не находится в пределах символьной константы, строкового литерала или комментария (вложенные комментарии недопустимы). Такой комментарий представляет собой последовательность многобайтных символов и заканчивается последовательностью */:

```
/*#include <stdio.h>
int main() {
   int a, b, c;
   scanf("%d%d%d", &a, &b, &c);
   printf("a = %d\nb = %d\nc = %d\n", a, b, c);
   return 0;
}*/
```

Второй тип – однострочный комментарий в стиле C++ – чтобы превратить строку в комментарий, нужно поместить в ее начало последовательность //

и завершить строку символом новой строки (комментарий не включает символ новой строки):

```
if (a > b && a > c) max = a; //первое число больше двух других
```

Однострочные комментарии могут быть вложенными:

```
//a=max_in_row(matr,rows,cols,rowindex);//поиск максимума в строке
```

2.2.4. Контрольные вопросы и упражнения

Контрольные вопросы

1. Какие множества символов используются в языке Си?
2. Какие буквы латинского алфавита относятся к базовым символам?
3. Для чего применяются управляющие последовательности символов? Какие из них входят в число базовых символов?
4. Перечислите стандартные целые типы со знаком.
5. Перечислите стандартные целые типы без знака.
6. Как соотносятся по количеству занимаемых в памяти байтов переменные разных целых типов?
7. Перечислите типы, применяемые для действительных чисел.
8. Перечислите типы, применяемые для комплексных чисел.
9. Из чего состоит объявление переменной? Приведите пример.
10. Какими бывают целые константы?
11. Что может быть у целой константы – суффикс или префикс? На что он указывает?
12. Как записывается десятичная целая константа?
13. Как записываются восьмеричная и шестнадцатеричная целые константы?
14. Как записывается десятичная константа с плавающей точкой?
15. Как записывается шестнадцатеричная константа с плавающей точкой?
16. Что такое символьная константа?

17. Что задают стандарты кодирования? Что такое ASCII?
18. Какие средства содержит стандарт Unicode для представления символов?
19. Что может содержать символьная константа - суффикс или префикс, на что он указывает?
20. Что такое строковый литерал, какой у него формат?
21. Существуют ли префиксы или суффиксы кодировки для строковых литералов?
22. Возможно ли объединение смежных строковых литералов, в каких случаях?
23. В каких целях нельзя применять ключевые слова?
24. Для чего предназначены знаки пунктуации?
25. С какой целью могут быть использованы диграфы, триграфы?
26. Как создать многострочный комментарий?
27. Как создать однострочный комментарий?
28. Какие комментарии могут быть вложенными?

Упражнения

Во всех перечисленных ниже упражнениях значения входных переменных, если они есть, нужно вводить с клавиатуры, а результаты – выводить на экран.

1. Напишите программу, которая выдает размер (в байтах) переменной типа char, short, int, long, float, double.
2. Напишите программу для вычисления длины окружности. Задайте число π в виде десятичной константы с плавающей точкой, радиус - типа double. Результат выведите на экран.
3. Напишите программу для вычисления площади круга. Задайте число π в виде шестнадцатеричной константы с плавающей точкой типа float, радиус – типа float. Результат выведите на экран.

4. Напишите программу для вычисления периметра треугольника. Стороны треугольника задайте в виде трёх десятичных констант типа long double. Результат выведите на экран.
5. Напишите программу, вычисляющую сумму двух шестнадцатеричных констант типа float. Результат выведите на экран.
6. Напишите программу, содержащую объявления трех символьных строковых литералов: наименования учебного заведения, ФИО студента, наименование изучаемой дисциплины, а также двух констант типа unsigned long long int: номер студенческого билета, номер зачётной книжки, и выведите значений этих констант на экран (для вывода значений целых констант используйте %ll или %llu).
7. Напишите программу, содержащую объявления трех расширенных строковых литералов: наименование учебного заведения, ФИО студента, наименование изучаемой дисциплины, а также двух десятичных констант типа double: средний балл аттестата, рейтинг, и выведите значения этих констант на экран (для вывода значений вещественных констант используйте %e).
8. Перепишите программу для упражнения №2 из п.2.1.3, используя диграфы всюду, где возможно.
9. Перепишите программу для упражнения №3 из п.2.1.3, используя триграфы всюду, где возможно.
10. Добавьте однострочные комментарии к программе для упражнения №6.

2.3. Операции

В языке Си имеются следующие группы операций [5]:

- арифметические операции;
- логические операции;
- операции сравнения;

- присваивания;
- инкремент/декремент;
- операции доступа к элементу;
- другие операции.

2.3.1. Особенности выполнения операций

Арифметические операции выполняют стандартные математические действия над своими операндами (табл. 3). Операции над беззнаковыми целыми числами выполняются по модулю 2^n, где n – количество разрядов беззнакового целого. Например, для unsigned int прибавление 1 к UINT_MAX (максимально возможное число данного типа) дает 0, а вычитание 1 из 0 дает UINT_MAX. Для целых чисел со знаком результат при переполнении не определен и зависит от реализации.

Таблица 3

Арифметические операции

Операция	Наименование	Пример
+	унарный плюс	+x
-	унарный минус	-x
+	сложение	x + y
-	вычитание	x - y
*	умножение	x * y
/	деление	x / y
%	остаток от деления	x % y
~	побитовое «не»	~x
&	побитовое «и»	x & y
|	побитовое «или»	x | y
^	побитовое исключающее или (xor)	x ^ y
<<	побитовый сдвиг влево	x << y
>>	побитовый сдвиг вправо	x >> y

Результат операции xor – 1, если операнды (биты) разные, и 0 в противном случае, например, 0 xor 1 = 1, 1 xor 1 = 0. Побитовый сдвиг влево выполняется как сдвиг влево битов операнда, находящегося слева от знака операции <<, на количество разрядов, заданных правым операндом. Освободившиеся разряды левого операнда заполняются нулями. Побитовый сдвиг вправо осуществляется аналогично, но биты сдвигаются вправо.

Логические операции выполняют стандартные булевы операции над своими операндами (табл. 4). Результаты этих операций имеют тип int, а операнды должны быть выражениями скалярных типов.

Таблица 4

Логические операции

Операция	Наименование	Пример
!	логическое «не»	!x
&&	логическое «и»	x && y
||	логическое «или»	x || y

Если значение выражения равно нулю, результат операции логического «не» – 1, в противном случае – 0. Правый операнд логического «и» вычисляется только в том случае, если значение левого не равно нулю. Результат операции логического «и» равен 1, если оба операнда не равны 0, в противном случае результат – 0. Правый операнд логического «или» вычисляется только в том случае, если значение левого равно нулю. Результат операции логического «или» равен 0, если оба операнда равны 0, в противном случае результат – 1.

Операции сравнения – это бинарные операции, которые проверяют условие и возвращают 1, если это условие истинно, и 0, если это условие ложно (табл. 5).

Результат каждой из операций сравнения имеет тип int, операнды могут быть действительного типа или указателями (указатель – объект, который хранит адрес другого объекта). Если операнды какой-либо из операций отношения >, <, >=, <= имеют действительный тип, то вначале над ними выполняются

обычные арифметические преобразования [5], а затем – сама операция, при этом положительный и отрицательный нули считаются эквивалентными, и любые сравнения с бесконечностью дают в результате нуль.

Таблица 5

Операции сравнения

Операция	Наименование	Пример
= =	равно	x == y
!=	не равно	x != y
>	больше	x > y
<	меньше	x < y
>=	больше или равно	x >= y
<=	меньше или равно	x <= y

В операциях проверки на равенство (= =) и неравенство (!=) оба операнда могут иметь арифметический тип (в том числе, complex и imaginary), при этом действия те же, что и для действительных операндов операций >, <, >=, <=. Подробности о сравнении операндов-указателей см. в [5].

Присваивания представлены в табл. 6.

Таблица 6

Присваивания

Операция	Наименование присваивания	Пример	Аналог
=	основное	x = y	нет
+=	со сложением	x += y	x = x + y
-=	с вычитанием	x -= y	x = x - y
*=	с умножением	x *= y	x = x * y
/=	с делением	x /= y	x = x / y
%=	со взятием остатка от деления	x %= y	x = x % y
&=	с побитовым «и»	x &= y	x = x & y
|=	с побитовым «или»	x |= y	x = x | y
^=	с побитовым исключающим «или»	x ^= y	x = x ^ y
<<=	с побитовым сдвигом влево	x <<= y	x = x << y
>>=	с побитовым сдвигом вправо	x >>= y	x = x >> y

Простое или основное присваивание x = y состоит в том, что значение y записывается в x, при этом старое значение x теряется. В сложных присваиваниях, которые перечислены в табл. 6, начиная с операции +=, над старым зна-

чением x и значением y выполняется операция, знак которой указан перед знаком = соответствующей операции (например, + в +=), результат записывается в левый аргумент (в табл. 6 левый аргумент – это x).

Операции инкремента/декремента – унарные операции, которые увеличивают/уменьшают соответственно на 1 значение переменной. Они имеют префиксную и постфиксную формы. В префиксной форме инкремент/декремент переменной i записываются соответственно так: ++i/--i, в постфиксной – i++/i--. Операндом указанных операций является изменяемое выражение, связанное с ячейкой памяти (modifiable lvalue), – т.е. нечто, куда можно записать значение, имеющее целый или один из вещественных типов (кроме комплексных), или являющееся указателем. Префиксная форма предполагает выполнение вначале инкремента/декремента, а затем остальных операций в операторе. При использовании постфиксной формы инкремент или декремент выполняется в последнюю очередь. Например:

```
int x = 1;
int y = ++x; //вначале ++x, затем y = x, т.е. в x – 2, в y – 2
y = x++; //вначале y = x, затем x++, т.е. в y – 2, в x – 3
```

Операции имеют приоритет, т.е. место в иерархии, определяющей порядок их применения к операндам в выражении. На вопрос, какая из операций, имеющих равный приоритет, будет выполнена первой, отвечают правила ассоциативности. Ассоциативность может быть правосторонней (выполнение операций справа налево) или левосторонней (слева направо). Сведения о приоритете и ассоциативности рассмотренных выше операций представлены в табл. 7.

Таблица 7

Приоритет и ассоциативность операций в языке Си

Приоритет	Операция	Описание	Ассоциативность
1	++ --	Постфиксные инкремент и декремент	Слева направо
	()	Вызов функции	
	[]	Обращение по индексу к элементу массива	
	.	Доступ к элементу структуры и объединения	
	->	Доступ к элементу структуры и объединения через указатель	
	(тип){список}	Составной литерал	
2	++ --	Префиксные инкремент и декремент	Справа налево
	+ -	Унарные плюс и минус	
	! ~	Логическое «не» и побитовое «не»	
	(тип)	Преобразование типа	
	*	Разыменование	
	&	Получение адреса	
	sizeof	Получение размера в байтах	
	_Alignof	Требования к выравниванию	
3	* / %	Умножение, деление, взятие остатка от деления	Слева направо
4	+ -	Сложение и вычитание	
5	<< >>	Побитовые сдвиги влево и вправо	
6	< <=	Для операций отношения < и <= соответственно	
	> >=	Для операций отношения > и ≥ соответственно	
7	== !=	Для операций отношения == и != соответственно	
8	&	Побитовое «и»	
9	^	Побитовое исключающее «или»	
10	|	Побитовое «или»	
11	&&	Логическое «и»	
12	||	Логическое «или»	
13	?:	Тернарная условная операция	Справа налево
14	=	Простое присваивание	
	+= -=	Присваивание с суммированием и разностью	
	*= /= %=	Присваивание с умножением, делением и остатком от деления	
	<<= >>=	Присваивание с побитовыми сдвигами влево и вправо	
	&= ^= |=	Присваивание с побитовыми «и», исключающим «или» и «или»	
15	,	Запятая	Слева направо

2.3.2. Контрольные вопросы и упражнения

Контрольные вопросы

1. Перечислите группы операций, которые имеются в языке Си.
2. Напишите знаки и назовите арифметические операции.
3. Как выполняются побитовые операции &, |, ~, xor?
4. Как выполняются побитовые сдвиги влево и вправо?
5. Перечислите логические операции, чему равны результаты их выполнения в зависимости от значений операндов?
6. Перечислите операции сравнения, назовите возможные типы операндов операций сравнения.
7. Что такое простое присваивание, как оно выполняется?
8. Поясните, как выполняются сложные присваивания, приведите примеры сложных присваиваний.
9. Как записываются и выполняются операции инкремента/декремента?
10. Чем отличаются префиксные инкремент/декремент от постфиксных?
11. Приоритет каких операций выше: постфиксных инкремента/декремента или префиксных инкремента/декремента?
12. Какая операция имеет самый низкий приоритет?
13. Какие операции имеют более высокий приоритет: сравнения или присваивания?
14. Какие операции имеют более высокий приоритет: унарные плюс/минус или бинарные плюс/минус?
15. Для чего логические операции «и», «или» имеют более низкий приоритет, чем сравнения?

Упражнения

1. Вычислите значение выражения: 65 | 13 << 2. Здесь и в аналогичных заданиях, приведенных ниже, нужно показать подробное выполнение всех операций с числами в двоичной системе.
2. Вычислите значение выражения: (121 >> 6) ^ 23.
3. Вычислите значение выражения: 37 << 3 & 29 >> 1.
4. Вычислите значение выражения: 89 >> 5 | 25 ^ 42.
5. Вычислите значение выражения: 112 >> 3 >> 2 << 5.
6. Обозначьте с помощью круглых скобок порядок применения операций к операндам в выражении: ++i + j++.
7. Обозначьте с помощью круглых скобок порядок применения операций к операндам в выражении: a >= b && a < c || e.
8. Обозначьте с помощью круглых скобок порядок применения операций к операндам в выражении: a += b -= c *= e.
9. Обозначьте с помощью круглых скобок порядок применения операций к операндам в выражении: c= a = c | e & b ^ a.
10. Обозначьте с помощью круглых скобок порядок применения операций к операндам в выражении: ~a == c || e != b.

3. Операторы

Здесь используется терминология, определяющая оператор как команду программы на языке Си или совокупность команд (составной оператор). Основные операторы языка Си:

1) составной оператор;

2) оператор вычисления выражения (оператор-выражение);

3) операторы выбора;

4) операторы циклов;

5) операторы переходов.

3.1. Составной оператор (блок) и оператор-выражение

Составной оператор или блок – заключённая в фигурные скобки последовательность операторов (включая декларации):

{

[<оператор1>

<оператор2>

…

<операторn>]

}

Составной оператор применяется как единый оператор там, где это требуется по смыслу. Например, при истинности некоторого условия может быть выполнена последовательность действий, объединённых в блок:

```
if (<условие>)
{ // начало блока
int a, b, c;  //объявление переменных
scanf("%d%d", &a, &b); // ввод a, b
c = a > b ? a : b;//если a > b, c присвоить a, иначе c присвоить b
printf("%d\n", c);//вывод c
} // конец блока
```

Тело любой функции в языке Си является составным оператором.

Оператор-выражение – это выражение, которое заканчивается точкой с запятой. Вызовы функций, присваивания также относят к операторам-выражениям. Оператор-выражение, не содержащий выражения, называется пустым оператором, это просто символ «;». Он нужен, чтобы обозначить пустое тело цикла, поставить метку в конце блока или перед декларацией. Примеры операторов-выражений:

```
1) my_func(param); //вызов функции my_func с аргументом param
2) x1 = (-b + sqrt(d))/(2*a); //вычисление значения выражения
```

3.2. Оператор if

Программы, предназначенные для решения практических задач, редко бывают линейными. Часто возникают ситуации, когда требуется выполнять какую-либо одну последовательность операторов в случае истинности[9] некоторого условия, а в случае его ложности – другую последовательность операторов. Т.е. программа разветвляется – разделяется на две ветви, начиная с некоторого действия. В общем случае ветвей может быть несколько. Для реализации ветвлений применяются операторы выбора, которые позволяют выбирать один из нескольких операторов (одну из нескольких ветвей программы) в зависимости от выполнения заданного условия. Условие в языке Си задается в виде контрольного выражения в скобках. Существует два типа операторов выбора: if и switch. Оператор if используется для выбора одного из двух вариантов, switch – для выбора из нескольких вариантов.

Оператор if имеет следующие формы синтаксиса:

a) if (<выражение>) <оператор>
b) if (<выражение>) <оператор1> else <оператор2>

В процессе выполнения оператора if вначале вычисляется значение выражения. Затем: в случае a, если значение выражения истинно, выполняется оператор, в противном случае оператор пропускается; в случае b, если значение выражения истинно, выполняется оператор1, в противном случае – оператор2. После этого управление переходит к следующему оператору программы, если в теле if не содержится оператор break или goto. Оператор if завершается символом «;» (это окончание простого оператора, входящего в состав if) или закрывающейся фигурной скобкой (это окончание составного оператора), например (вторая форма с простыми операторами):

```
if (a > b)
```

[9] В языке Си нуль – это ложь, а все, что не нуль – истина. Например, результат выражения, равный 1 – истина, 10 – истина, -54 – истина, 0 – ложь.

```
    printf("a > b\n");
else
    printf("a <= b\n");
```

Для составных операторов обязательно, а при вложениях желательно использовать фигурные скобки. Если есть вложения, каждое ключевое слово else связывается с ближайшим оператором if, который не имеет else. Например, блок кода

```
if (a > b)
   if (a > c)
       printf("a > b и a > c\n");
   else
       printf("a > b и a <= c\n");
else
   if (b > c)
      printf("b >= a и b > c\n");
   else
      printf("b >= a и b <= c\n");
```

нагляднее представить следующим образом:

```
if (a > b) {
   if (a > c) {
       printf("a > b и a > c\n");
   }
   else {
       printf("a > b и a <= c\n");
   }
}
else {
   if (b > c) {
      printf("b >= a и b > c\n");
   }
   else {
      printf("b >= a и b <= c\n");
   }
}
```

3.3. Операторы goto и switch

Любой оператор (кроме декларации – в языке Си) может быть помечен с помощью метки, которая ставится перед ним с последующим двоеточием:

<имя_метки> : <оператор>

Метка является идентификатором, который должен быть уникальным в пределах функции, в которой она задана. Метка сама по себе ни на что не влия-

ет, в частности, не изменяет последовательность выполнения операторов. Она нужна для того, чтобы на помеченный ею оператор можно было перейти из другой точки программы (с помощью операторов, обеспечивающих такой переход – goto или switch).

Оператор goto вызывает безусловный переход на оператор, помеченный указанной меткой. Оператор применяется в случаях, когда по-другому невозможно передать управление в нужное место[10]. Синтаксис:

goto <имя_метки>;

…

<имя_метки> : <оператор>

Переход в область действия массива переменной длины или другого модифицируемого типа невозможен:

```
 int n = 7;
//goto lbl;//ошибка: метка указана после декларации массива v
 int v[n]; // массив переменной длины
 lbl:   int i;
```

Оператор switch предназначен для перехода на один из нескольких помеченных операторов. Синтаксис:

switch (<выражение>) {

  case (<константное_выражение_1>) : <оператор1>

  case (<константное_выражение _2>) : <оператор2>

  …

  case (<константное_выражение _N>) : <операторN>

  default : <операторN+1>

}

В представленном выше описании выражение должно иметь целочисленный результат, константные выражения в круглых скобках после ключевого слова case также преобразуются в целые константы и являются case-метками

[10] В связи с тем, что управление в нужное место практически всегда можно передать другим способом, оператор goto применять не рекомендуется.

соответствующих операторов. Если значение выражения, указанного в круглых скобках после ключевого слова switch, оказывается равным значению одного из константных выражений, происходит переход на оператор, помеченный этим константным выражением. Если значение исходного выражения не совпало ни с одним из константных выражений, и в теле switch есть метка default, то происходит переход на оператор, помеченный этой меткой. Если нет совпадения и нет метки default, тело switch не выполняется.

На рис. 3 показано, что оператор switch осуществляет переход на один из помеченных операторов его тела (которое заключено в фигурные скобки), но затем выполняются все нижеследующие операторы его тела, если не был выполнен принудительный переход в какое-либо другое место с помощью одного из операторов break (выход из тела switch), goto, return.

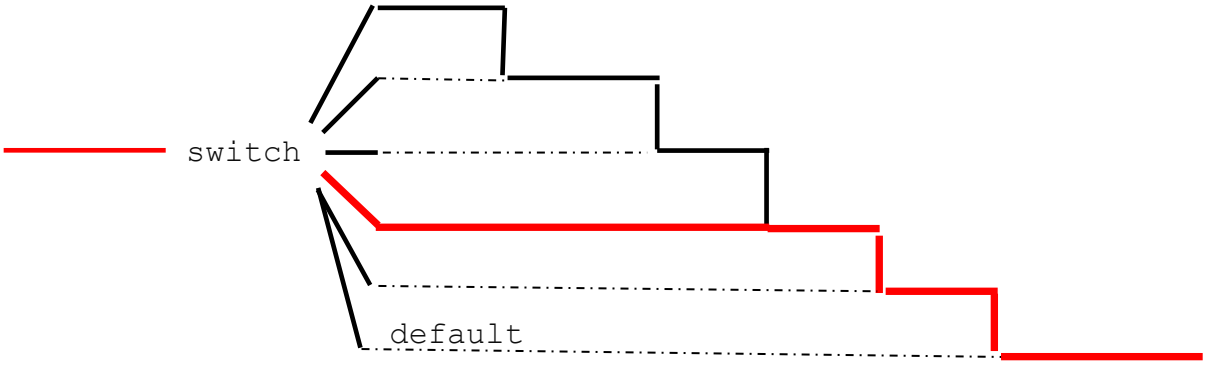


Рис. 3. Иллюстрация перехода на помеченный оператор в switch

Массив переменного размера может быть объявлен либо до оператора switch, либо после последней метки.

3.4. Операторы циклов, операторы break и continue

В программах часто требуется выполнять одну и ту же последовательность действий несколько раз. Пусть, например, требуется вычислить квадратные корни из всех натуральных чисел заданного диапазона и вывести на экран числа и вычисленные значения. Соответствующую программу для диапазона 11-13 можно записать так:

```
#include <stdio.h>
#include <math.h>
int main() {
  unsigned value;
  double result;
  value = 11;
  result = sqrt((double)value);
  printf("%u  %.3lf\n", value, result);
  value = 12;
  result = sqrt((double)value);
  printf("%u  %.3lf\n", value, result);
  value = 13;
  result = sqrt((double)value);
  printf("%u  %.3lf\n", value, result);
  return 0;
}
```

Повторяющийся в этой программе блок операторов (5),

```
value = 11;
result = sqrt((double)value);
printf("%u  %.3lf\n", value, result);
```
(5)

где переменной value последовательно присваивались значения 11, 12, 13, записан 3 раза. Если диапазон будет состоять не из трех, а, например, из 100000 значений, длина программы резко увеличится. Если же диапазон чисел будет вводиться пользователем во время работы программы, записать её вышеуказанным способом не получится, так как заранее неизвестно, с какого числа начинать вычисления и сколько раз их нужно повторить. Поэтому для реализации многократных повторений применяются специальные, циклические конструкции, позволяющие саму последовательность действий записать один раз и указать, сколько раз она будет повторяться, для каких данных, какие изменения нужно выполнять над данными. Подлежащая многократному выполнению последовательность операторов (блок в фигурных скобках или один простой оператор) называется телом цикла. Однократное выполнение тела цикла является итерацией цикла.

Какие наиболее простые и понятные действия можно выполнить, используя циклическую конструкцию? Можно, например, накопить сумму целых чисел, которые вводит пользователь. Пусть сумма хранится в переменной s, а вво-

димые числа помещаются в переменную x, каждый раз при вводе предыдущее число теряется. Тело цикла будет выглядеть так:

```
{
   scanf("%d", &x);
   s = s + x;
}
```

Перед началом выполнения итераций в переменную s следует записать 0 (инициализировать s нулем). В операторе s = s + x; вначале вычисляется правая часть относительно знака «=», т.е. берется «старое» значение s, к нему прибавляется значение x. Затем в s записывается эта новая сумма, а «старое» значение s теряется. Поэтому если оператор s = s + x; выполнится несколько раз, в переменной s накопится сумма чисел, введенных пользователем.

3.4.1. Операторы while, do .. while, for, break, continue

В языке Си есть три типа операторов цикла: while, do … while и for.

Оператор while

Синтаксис:

```
while (<выражение>) <тело_цикла>
```

Здесь и далее <тело_цикла> – это оператор, простой или составной (в последнем случае тело цикла помещается в фигурные скобки). Оператор while повторяет выполнение оператора, составляющего его тело, до тех пор, пока не станет ложным условие, заданное выражением в круглых скобках после ключевого слова while.

Выполнение цикла while, при отсутствии в теле while операторов goto, return, break, состоит из следующих шагов:

1. Вычислить значение выражения. Если результат – ложь, перейти на шаг 3, иначе – на шаг 2.
2. Выполнить тело цикла, перейти на шаг 1.

3. Перейти на оператор в программе, непосредственно следующий за телом while.

Таким образом, вначале вычисляется значение выражения и, в случае его истинности, выполняется тело цикла, после чего вновь вычисляется значение выражения, в случае его истинности вновь выполняется тело цикла и т.д. до тех пор, пока оно не станет ложным. Если в теле while есть один из операторов goto, return, break, возможен досрочный выход из цикла.

В качестве примера далее рассматривается задача вычисления факториала (!) неотрицательного целого числа n: n! = 1·2·3·…·n, 0! = 1, 1! = 1, 2! = 1·2 = 2, 3! = 1·2·3 = 6 и т.д. Потребуется одна целочисленная переменная (счетчик цикла) для задания условия и выполнения промежуточных вычислений, которая будет принимать значения от 1 до n с шагом 1:

```
int i = 1;
```

Можно задать следующее условие (выражение в скобках после ключевого слова while):

```
i <= n
```

Для хранения промежуточных и окончательного результатов объявляется еще одна переменная, ей присваивается начальное значение 1:

```
int result = 1;
```

Если выполнять составной оператор

```
{ result = result*i;  i = i + 1; }
```

в качестве тела цикла, то получается, в частности, для n = 4:

```
result = 1*1; i = 1+1; //1 итерация
result = 1*2; i = 2+1; //2 итерация
result = 2*3; i = 3+1; //3 итерация
result = 6*4; i = 4+1; //4 итерация
```

Следующей итерации не будет, так как значение i равно 5, и условие стало ложным. Для n = 0 цикл не выполнится ни разу, и в переменной result будет верный результат (1), а для n = 1 выполнится лишь первая итерация, результат также будет верным. Теперь можно записать конструкцию целиком:

```
int n = 4, i = 1, result = 1;
while (i <= n) {
   result = result*i;
   i = i + 1;
}
```

Оператор do .. while

Синтаксис:

```
do <тело_цикла>
while (<выражение>)
```

Это цикл с постусловием, в нем вначале выполняется тело цикла, а затем проверяется условие, которое задается выражением в круглых скобках после ключевого слова while. Если условие (значение выражения) истинно, то тело цикла выполняется вновь. Так происходит до тех пор, пока условие не станет ложным. Таким образом, в цикле с постусловием тело цикла выполнится, как минимум, один раз – в том случае, когда условие изначально было ложным. Выполнение цикла do .. while, при отсутствии в теле операторов goto, return, break, состоит из следующих шагов:

1. Выполнить тело цикла, перейти на шаг 2.
2. Вычислить значение выражения. Если результат – ложь, перейти на шаг 3, иначе – на шаг 1.
3. Перейти на оператор в программе, непосредственно следующий за конструкцией do .. while.

Оператор for

Цикл for в простом случае позволяет выполнять некоторую последовательность действий заданное число раз. Для этого в нем используется специальная переменная целого типа, которая называется индексом (счетчиком) цикла. Эта переменная последовательно принимает все значения из заданного диапазона с определенным шагом. Если очередное значение счетчика цикла удовлетворяет заданному условию, т.е. находится в пределах диапазона, выполняется тело цикла. Цикл перестанет выполняться сразу после того, как значение счетчика цикла выйдет за границы указанного диапазона. В общем случае конструкция for позволяет использовать множество переменных, которые могут

изменяться от итерации к итерации, и компактно кодировать нетривиальные алгоритмы.

Синтаксис:

```
for (<инициализация - 1>;<условие - 2>;<изменение переменных - 4>)
    <тело цикла - 3>
```

Тело цикла, как и ранее, представляет собой один простой оператор или несколько операторов, т.е. блок. Блок необходимо заключить в фигурные скобки. В разделе «инициализация» однократно, в самом начале работы цикла, выполняются присваивания начальных значений переменным, которые используются в цикле. Это может быть инициализация ранее объявленных переменных, например:

```
i = 0, j = 10;
```

или определения переменных, которые видимы только в данном цикле, например:

```
int k = 56, r = -2, i = 0;
```

В разделе «условие» задается условное выражение, значение которого, истина или ложь, вычисляется перед очередным выполнением тела цикла для проверки возможности последующего выполнения тела цикла. Если условие составное, его части связываются с помощью операций логического «и» - &&, логического «или» - || и т.д. Например:

```
 i < 100 && j > 0;
```

В разделе «изменение переменных» записывается один или несколько операторов, которые выполняются после каждой итерации – после каждого выполнения тела цикла. Например:

```
i = i + 2, j --;
```

Условное выражение вычисляется в первый раз после блока инициализации. Затем, если его значение истинно, выполняется первая итерация цикла, т.е. тело цикла. После этого выполняется блок изменения переменных. Затем опять вычисляется значение условного выражения, и т.д. до тех пор, пока значение условного выражения не станет ложным. Таким образом, имеет место следую-

щий порядок выполнения операторов, реализуемых оператором for: вначале выполняется инициализация – 1, затем проверяется условие – 2; если условие истинно, выполняется тело цикла – 3, далее выполняется изменение переменных – 4 и снова – 2, если истинно – 3, затем – 4, и т.д. до тех пор, пока условие не станет ложным. Как только условие станет ложным, произойдет переход на оператор, находящийся непосредственно за телом цикла (выход из цикла).

Блоки операторов в круглых скобках после for не являются обязательными. Можно, например, писать так:

```
for (; i < 10; ) { … }
```

или так:

```
for (; ;) { … }
```

В последнем примере цикл является бесконечным.

Любой цикл может не иметь тела, в таком случае вместо тела пишется пустой оператор – точка с запятой:

```
while (<условие>)
    ;

do
  ;
while (<условие>);

for (i = 0; i < 10; ++i)
   ;
```

Оператор break применяется для досрочного выхода из циклов while, do .. while, for (а также из тела оператора switch). Можно выйти по break-у из тела цикла, если выполнилось определенное условие, или задать бесконечный цикл, из которого будет осуществляться выход по break-у в результате выполнения условия. В частности, оператор

```
for (; i < 100; ++i) { <операторы> }
```

эквивалентен оператору

```
for (;; ++i) {
    if (i >= 100) break;
    <операторы>
}
```

Если нужно прервать выполнение текущей итерации цикла и перейти к следующей, применяется оператор continue. В операторе for он передает управление в блок «изменение переменных». Пусть требуется вычислить сумму только положительных чисел из n целых чисел, введенных пользователем. Соответствующий фрагмент программы с оператором continue можно записать так:

```
int x, n, s = 0;
scanf("%d", &n);
for (int i = 1; i <= n; ++i) {
  scanf("%d", &x);
  if (x <= 0) continue;
  s = s + x;
}
```

Если введенное значение x меньше или равно нулю, управление передается на оператор ++i по continue. Поэтому оператор s = s + x; выполняется только в случае x > 0.

Выполнившийся в теле цикла while или do .. while оператор continue передает управление на оператор вычисления условного выражения, записанного в круглых скобках после ключевого слова while.

3.4.2. Кое-что об отладке

Почти невозможно научиться программировать, не программируя самостоятельно, не закрепляя полученные на занятиях по программированию знания выполнением домашней работы. В процессы алгоритмизации и кодирования имеет смысл вовлекаться постепенно. Чтобы научиться исправлять синтаксические ошибки, следует внимательно читать соответствующие пояснения (возможно, на английском языке, используя при необходимости переводчик): может отсутствовать точка с запятой в конце оператора, недоставать фигурной скобки, часто допускаются ошибки в количестве или типах аргументов функций и т.п. Если что-то в данный момент «не работает», это не так плохо, как может показаться на первый взгляд. Часто «не работает» и «не понимаю, что

написано» оказываются синонимами. Есть два вида действий, весьма полезных при разработке программ:

– продумывание алгоритмов, решение нескольких задач на одну тему;

– отладка каждой программы (выполнение по шагам, просмотр значений переменных) до тех пор, пока задача не будет решена для всех возможных классов входных данных или хотя бы – пока не станет понятна логика работы программы.

Тем, кто пока еще способен только изучать готовые программы и немного изменять их, важно знать: переход из состояния непонимания в состояние понимания может произойти в любой момент, внезапно. Нужно накопить критическую массу количественных сведений, пусть еще не до конца понятных, для того чтобы этот переход смог состояться. Теоретический материал следует читать внимательно, систематически и, возможно, многократно.

Особенно важно понять, как работают циклы. Для этого необходимо освоить порядок отладки программ. Отладка предполагает пошаговое выполнение программы с целью обнаружения точек (операторов), в которых она неправильно работает. В Visual Studio.Net пошаговое выполнение программы без входа в подпрограммы выполняется с помощью нажатия на клавишу F10, со входом в подпрограммы – F11. Для выполнения всей программы, начиная с первого оператора main, по шагам после написания программного кода следует нажать F10, исправить, если есть, синтаксические ошибки, и вновь нажать F10. Возможно, эти действия придется повторить несколько раз – до тех пор, пока не будут исправлены все синтаксические ошибки. Когда программа запустится, можно будет увидеть стрелку (по умолчанию, желтую) слева от первого оператора функции main. При очередном нажатии F10, текущий оператор выполнится, после чего стрелка установится на следующем операторе. Если текущий оператор содержал вызов функции ввода, то программа приостановится: в окне ввода (консоли) нужно ввести необходимые данные, а затем переключиться вновь в окно, где программа выполняется по шагам и снова нажать F10. В про-

цессе такого пошагового выполнения можно подносить указатель мыши к переменной и видеть её значение. В частности, можно смотреть, как изменяется счетчик цикла от итерации к итерации. В Visual Studio.Net также есть специальное окно с вкладками «Видимые», «Автоматические», «Контрольные значения», в которых можно просматривать переменные во время отладки. В окно «Контрольные значения» можно добавлять новые переменные или удалять ненужные. Если требуется проанализировать пошаговую работу некоторой функции (подпрограммы), нужно нажать на соответствующем операторе, содержащем вызов этой функции, не F10, а F11. Произойдет переход на первый оператор, содержащийся внутри данной функции, и дальше следует нажимать F10, если не требуется выполнить вход в другую функцию, вызываемую уже из данной. После окончания выполнения текущей функции по шагам произойдет переход на первый оператор, записанный после оператора, содержащего вызов данной функции. Если выполнение по шагам больше не требуется, можно нажать F5. Также для остановки/прерывания отладки нужно нажать Shift+F5. Можно поставить точку останова на операторе, нажав F9, при этом слева от оператора появится красный кружок. Точка останова – это оператор, до которого программист не хочет выполнять программу по шагам, так как знает почти наверняка, что ошибок до этой точки нет. Поэтому вначале он запускает программу в обычном режиме отладки (нажимает F5), а затем, после того как точка останова достигнута, можно выполнить программу по шагам, нажимая F10 или F11, просматривая значения переменных. Если требуется временная точка останова, следует навести указатель мыши на строку кода, чтобы слева появилась кнопка «Выполнение до щелкнутого» (зеленая кнопка в виде треугольника и вертикальной черты), на неё нужно нажать. При этом часть кода между предыдущим оператором и данной временной точкой выполнится сразу (не пошагово). Использование временных точек останова ускоряет отладку: позволяет, например, пропускать большие участки кода между парами постоянных точек останова.

В других средах разработки, скорее всего, используются другие функциональные клавиши для отладки. Их можно посмотреть в соответствующем пункте меню (в пункте «Отладка» или «Debug» и т.п. той среды разработки, которая применяется для разработки программы). В самом плохом случае, когда по каким-либо причинам пошаговое выполнение затруднено, можно выводить промежуточные результаты вычислений на экран и по ним определять вероятные ошибки.

3.4.3. Вычисление суммы ряда с использованием оператора for

Пусть рассматривается ряд чисел $\frac{a_0}{b_0}, \frac{a_1}{b_1}, ..., \frac{a_n}{b_n}, ...$, где каждое последующее число может быть получено из предыдущего или из первого с помощью определенных действий, выполненных над числителем и знаменателем. Например, каждый последующий числитель может быть квадратом предыдущего, а каждый последующий знаменатель – i-й степенью знаменателя первого числа, где i – номер числа в ряде. Требуется найти сумму n первых чисел ряда. Общий алгоритм вычисления суммы обычно состоит в том, что к начальному значению суммы, которое равно нулю, последовательно прибавляются элементы ряда: вначале первый, затем второй и т.д. Пусть числитель хранится в переменной a, знаменатель – в b, сумма – в s. Сумма инициализируется нулем, начальные значения числителя и знаменателя вводятся или присваиваются:

```
s = 0, a = 3, b = 2;
```

Затем в цикле выполняется действие

```
s = s + a/b; //новое значение суммы – это предыдущая (старая) сумма плюс текущее число a/b
```

после которого числитель и знаменатель нужно изменить так, как требуется, например:

```
a = a*a; //новое значение числителя – квадрат старого значения
b = b*2; //новое значение знаменателя 2^i, i – номер числа
```

Теперь можно записать циклическую конструкцию (включая объявления переменных) полностью следующим образом:

```
int i, n = 4;
double a, b, s;
for (s = 0, a = 3, b = 2, i = 1; i <= n; ++i) {
     s = s + a/b;
     a = a*a;
     b = b*2;
}
```

или, переместив тело цикла в раздел изменения переменных в for и заменив простые присваивания на сложные, так:

```
for (s=0, a=3, b=2, i=1; i<=n; s+=a/b; a*=a; b*=2; ++i) ;
```

Иногда требуется чередовать знак перед элементами, например, вычислять такую сумму: $\frac{a_0}{b_0} - \frac{a_1}{b_1} + ... + (-1)^n \frac{a_n}{b_n}$. Один из способов реализации – включить в программу новую переменную, первоначально равную 1, и умножать на нее текущее слагаемое a/b, а саму переменную на каждой итерации умножать на -1:

```
int i, k = 1, n = 4;
double a, b, s;
for (s = 0, a = 3, b = 2, i = 1; i <= n; ++i) {
     s = s + k*a/b;
     a = a*a;
     b = b*2;
     k = -k;
}
```

Чтобы написанное стало еще более понятным, представленная ниже простая циклическая конструкция записывается далее в виде линейной последовательности операторов:

```
int i, n = 3, x = 2, s1 = 0, s2 = 0;
for (i = 1; i < n; ++i) {
     s1 = s1 + x;
     s2 = s2 – 3*x;
     x = x + i;
}
printf("%d\t%d\n", s1, s2);
```

Так выглядит этот фрагмент кода в «развернутом» виде:

```
int i, n = 3, x = 2, s1 = 0, s2 = 0;
i = 1;   //первый оператор цикла for
1 < 3;   //истина
```

```
s1 = 2;  //s1 = s1 + x;
s2 = -6; //s2 = s2 – 3*x;
x = 3;   // x = x + i;
i = 2;   // ++i;
2 < 3;   //i <= n;
s1 = 5;  //s1 = s1 + x;
s2 =-15; //s2 = s2 – 3*x;
i = 3;   //++i;
3 < 3;   //ложь
printf("%d\t%d\n", s1, s2); //первый оператор после for
```

3.4.4. Одномерные массивы и циклы

Ранее рассматривались данные простых типов: int, char, double. Операторы циклов часто применяются для обработки последовательно расположенных в памяти однотипных данных. Для хранения и обработки совокупности данных одного типа используются непрерывные участки памяти, логически разбитые на блоки одинакового размера. В каждом блоке размещается один элемент данных. Совокупность однотипных элементов, размещаемых последовательно в непрерывном участке памяти, называется массивом. Обычный статический массив имеет размер (количество элементов), имя (идентификатор); тип в его объявлении задает тип каждого элемента.

Объявление массива из 10 элементов типа int записывается так: int a[10];
Лучше объявить заранее константу, в которую записывается размер массива:

```
const int n = 10;
int a[n];
```

Традиционно обращение к элементу массива выполняется по индексу, указанному в квадратных скобках. Индекс – это номер элемента. В языке Си элементы массива нумеруются с нуля, т.е. первый элемент имеет индекс 0, второй – 1, и т.д., последний элемент имеет индекс n-1, где n – размер массива. Присваивание значений некоторым элементам массива выполняется так:

```
a[0] = 5; a[9] = -34;  //a[9] – последний элемент массива
```

Ввод элементов массива:

```
scanf("%d", &a[2]);  scanf("%d", &a[1]);  scanf("%d", &a[8]);
```

Элемент a[10] не существует, обращение к нему приведет к ошибке.

Далее рассматриваются простые операции, которые выполняются в цикле для одномерных массивов. Начинающим рекомендуется выполнить приведенные фрагменты кода пошагово, просматривая значения всех изменяющихся переменных. Ввод или генерацию всех элементов массива можно выполнить так:

```
for (int i = 0; i < n; ++i) {
  printf("a[%d] = ", i); //подсказка для ввода
  scanf("%d", &a[i]);    //ввод
  //a[i]=rand()%100;//генерация целого числа из диапазона 0 .. 99
}
```

Вычисление суммы значений элементов массива реализуют, последовательно прибавляя к текущей сумме очередной элемент массива:

```
int s = 0;
for (int i = 0; i < n; ++i) {
  s = s + a[i];
}
printf("Сумма элементов массива %d\n", s);
```

Вычисление суммы значений элементов, соответствующих условию, например суммы всех неотрицательных элементов, предполагает проверку условия для каждого элемента:

```
int s = 0;
for (int i = 0; i < n; ++i) {
  if (a[i] >= 0)
     s = s + a[i];
}
printf("Сумма неотрицательных элементов массива %d\n", s);
```

Для обмена значений элементов с индексами i и j можно ввести дополнительную переменную temp:

```
int temp = a[i];
a[i] = a[j];
a[j] = temp;
```

Поиск элемента с минимальным значением, в соответствии с алгоритмом, приведенным в начале данного пособия:

```
int min = a[0];
```

```
for (int i = 1; i < n; ++i) {
  if (a[i] < min)
     min = a[i];
}
```

Простое упорядочивание (сортировка) элементов по возрастанию предполагает сравнение первого элемента со 2, 3, … n-м, сравнение второго – с 3, 4, …, n-м, …, сравнение n-1-го – с n-м. Если при каком-либо сравнении элементы не расположены в порядке возрастания, они меняются местами:

```
int i,j, temp;
for (i = 0; i < n-1; ++i) {
  for (j = i+1; j < n; ++j) {
     if (a[i] > a[j]) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
     }
  }
}
```

Переписывание элементов из одного массива в другой по какому-либо условию должно учитывать, что в разных массивах используются разные переменные для индексов. Запись в массив b всех отрицательных элементов из a:

```
int b[n], j = 0;
for (int i = 0; i < n; ++i) {
  if (a[i] < 0) {
    b[j] = a[i];
    j = j + 1;
  }
}
```

Можно заметить, что после окончания цикла размер массива b содержится в j: например, если в a 2 отрицательных элемента, то на выходе из цикла в j записано 2 (a элементы b имеют индексы 0 и 1), если в a 3 отрицательных элемента, то в j – 3 и т.д.

3.4.5. Вложенные циклы

На практике часто возникает необходимость организовывать один цикл внутри тела другого, внешнего цикла. Такой внутренний цикл называется вложенным. Рассмотрим распространенный алгоритм сортировки элементов одно-

мерного массива пузырьком. Сортировка выполняется в несколько проходов. Проход – просмотр массива от первого до последнего элемента. На каждом проходе проверяется, нарушен ли порядок расположения каждой пары соседних элементов. Если нарушен, то эти соседние элементы меняются местами.

Пусть, например, дана следующая последовательность элементов, которую нужно отсортировать по возрастанию значений: -1 4 3 -4 6 7 -10. В результате первого прохода поменяются местами элементы 4 и 3, 4 и -4, 7 и -10, и последовательность станет такой: -1 3 -4 4 6 -10 7. На втором проходе (вновь от начала) поменяются местами элементы 3 и -4, 6 и -10: -1 -4 3 4 -10 6 7. Последовательность после третьего, четвертого, пятого и шестого проходов, соответственно, будет такая: -4 -1 3 -10 4 6 7; -4 -1 -10 3 4 6 7; -4 -10 -1 3 4 6 7; -10 -4 -1 3 4 6 7. Максимальное количество проходов по массиву равно числу его элементов. В рассматриваемом случае будет выполнен еще один проход, на котором не произойдет ни одного обмена, так как последовательность уже отсортирована.

Для реализации алгоритма потребуется внешний цикл, каждая итерация которого соответствует одному проходу. На каждом проходе должен выполняться внутренний цикл, в котором сравниваются пары соседних элементов. В результате первого прохода на последней позиции оказывается максимальный элемент, в результате второго – на предпоследней позиции – максимальный из оставшихся элементов и т.д. Поэтому после каждого прохода количество сравниваемых во внутреннем цикле элементов (h) уменьшается на единицу:

```
const int n = 7;

int a[n], i, j, temp, h = n;
for (i = 0; i < n; ++i) {//для каждого i j изменяется от 0 до h-1
  for (j = 0; j < h-1; ++j) {
      if (a[j] > a[j+1]) { temp=a[j]; a[j]=a[j+1]; a[j+1]=temp; }
  }
  --h;
}
```

Вложенные циклы также применяются для обработки двумерных массивов. Двумерный массив представляется собой массив, элементами которого являются не числа, а массивы. Например, объявление

```
double mas[5][3];
```

задает массив из пяти элементов (строк), каждый из которых является массивом из трех элементов типа double (в каждой строке 3 числа). Т.е. указанное объявление задает матрицу действительных чисел размерностью 5×3. Для обращения к элементу в квадратных скобках после имени массива указываются номер строки и номер столбца (нумерация начинается с нуля, т.е. первая строка имеет номер 0, первый столбец – 0 и т.д.): mas[0][0] – первый элемент первой строки, mas[0][1] – второй элемент первой строки, …mas[1][2] – третий элемент второй строки и т.д.

Для ввода элементов матрицы применяется двойной вложенный цикл. Возможны два варианта: внешний цикл по строкам, внутренний – по элементам строки (по столбцам) или внешний цикл по столбцам, внутренний – по элементам столбца. Вот как выглядит первый вариант:

```
const int n = 5;
const int m = 3;
double mas[n][m];
int i, j;
for (i = 0; i < n; ++i)  {
   for (j = 0; j < m; ++j)  {
       printf("mas[%d][%d] = ", i, j);
       scanf("%lf", &mas[i][j]);
   }
}
```

3.5. Контрольные вопросы и упражнения

Контрольные вопросы

1. Перечислите основные операторы языка Си.
2. Что такое составной оператор (блок)?
3. Что такое оператор-выражение?

4. Что есть истина в контексте языка Си?
5. Что есть ложь в контексте языка Си?
6. Какие формы синтаксиса имеет оператор if?
7. Для чего предназначен оператор if, что происходит в результате его выполнения?
8. Назовите особенности применения ключевого слова else.
9. Для чего предназначен оператор goto, что происходит в результате его использования?
10. Для чего предназначен оператор switch, опишите его синтаксис.
11. Для чего в операторе switch применяется оператор break?
12. Представьте синтаксис и порядок выполнения цикла while.
13. Когда вычисляется условное выражение оператора цикла while, до или после тела цикла?
14. Представьте синтаксис и порядок выполнения цикла do .. while.
15. Чем цикл do .. while отличается от цикла while?
16. Представьте синтаксис и порядок выполнения цикла for.
17. Что происходит после вычисления условного выражения в цикле for (если оно есть)?
18. Для чего предназначен оператор break? Приведите примеры использования.
19. Для чего предназначен оператор continue? Приведите примеры использования.
20. Как выполнить программу на языке Си (C++) по шагам в среде Visual Studio?
21. Представьте общий алгоритм вычисления суммы ряда чисел, в котором каждое последующее число может быть получено из предыдущего или из первого с помощью определенных операций над числителем и знаменателем.
22. Что такое одномерный массив?

23. Как ввести элементы одномерного массива?
24. Как вычислить сумму или произведение элементов одномерного массива?
25. Что такое сортировка? Приведите простой алгоритм сортировки одномерного массива.
26. Что такое вложенный цикл?
27. Запишите и объясните алгоритм сортировки одномерного массива пузырьком.
28. Запишите и объясните алгоритм обработки элементов двумерного массива (матрицы) по строкам. Под обработкой можно понимать ввод, генерацию, вывод и другое.
29. Запишите и объясните алгоритм обработки элементов матрицы по столбцам.

Упражнения

Написать программу, решающую задачу:

1. Ввести 2 целых числа a и b. Вывести одно из текстовых сообщений «первое больше», «второе больше», «числа равны». В следующей строке вывести значение корня квадратного из модуля большего числа.
2. Ввести 3 целых числа. Если среди них есть хотя бы 2 равных, то увеличить все числа на 15 и вывести на экран. В противном случае выдать ответ «равных нет».
3. Ввести два целых числа x и y. Заменить x их произведением, а y – утроенной суммой, если x больше y, в противном случае x заменить разностью исходных чисел x и y, а y – их произведением.
4. Ввести натуральное число, интерпретируемое как оценка. Если введенное значение попало в диапазон [1, 5], вывести строку – словесное описание соответствующей оценки (1 – «плохо», 2 – «неудовлетворительно», 3 – «удовлетворительно», 4 – «хорошо», 5 –

«отлично»). Если введенное значение больше 5, вывести строку «ошибка ввода».

5. Ввести три целых числа x, y, z. Вывести наибольшее из значений |x-y|, |y-z|, |z-x|.
6. Ввести номер месяца (1 – январь, 2 – февраль, ...). Вывести название соответствующего времени года («зима», «весна», «лето» или «осень»).
7. Ввести целое число. Если оно попало в диапазон [0, 9], вывести строку – название соответствующей цифры на русском языке (0 – «ноль», 1 – «один», 2 – «два», ...), в противном случае вывести строку «не цифра».
8. Ввести два вещественных числа и знак операции, которую над ними необходимо произвести (+, -, /, *), выполнить соответствующую операцию над числами и вывести результат.
9. Для введенного года определить, является ли он високосным. Високосным является год, который делится без остатка на 4, за исключением случая, когда он делится на 100 и не делится на 400.
10. Ввести действительные числа x, y, z.

    Вычислить min{2x + y/3 + z/5, xy+z/3} + 1.
11. Если сумма трех введенных попарно различных действительных чисел x, y, z меньше трех, то наименьшее из этих чисел заменить полусуммой двух других; в противном случае заменить большее из y и z суммой двух оставшихся значений.
12. По введенным номеру квартиры, количеству квартир на лестничной клетке и количеству этажей в доме определить номер подъезда и этаж; вывести: подъезд, этаж, номер квартиры.
13. С помощью оператора switch и цикла do .. while создать меню для выбора одного из семи действий над двумя действительными числами: ввод, вывод, сумма, разность, частное, произведение, выход.

14. Ввести действительные числа a, b, c. Найти действительные корни квадратного уравнения $ax^2+bx+c=0$, если они существуют. Если действительных корней нет, вывести соответствующее сообщение.
15. Написать программу, которая просит пользователя вводить целые числа от 1 до 9 и складывает их до тех пор, пока общая сумма не превысит 100.
16. Ввести натуральное число n, действительные числа b_1, b_2, …, b_n. Вычислить $b_1^3 + b_2^3 + \ldots + b_n^3$.
17. Вывести таблицу температур по Цельсию (T°C) от 0 до 200 градусов и их эквивалентов по Фаренгейту (TF) с заданным пользователем шагом, используя формулу TF = 9/5*T°C + 32.
18. Ввести натуральное число n, целые числа b_1, b_2,…, b_n. Найти количество и сумму тех членов последовательности b_1, b_2, …, b_n, которые делятся без остатка на 5 и не делятся на 3.
19. Ввести натуральное число n. Вычислить $n!!/e^n$.
20. Ввести натуральное число n. Вывести количество различных цифр, встречающихся в его десятичной записи.
21. Вычислить сумму n первых членов ряда: $x, -\frac{1}{2}x^2, \frac{1}{3}x^3, -\frac{1}{4}x^4, \ldots$. Величины n и x ($|x| < 1$) вводятся пользователем.
22. Вычислить сумму n первых членов ряда: $1, x, \frac{x^2}{2!}, \frac{x^3}{3!}, \ldots$. Величины n и x вводятся пользователем.
23. Вычислить сумму n первых членов ряда: $x, -\frac{1}{3!}x^3, \frac{1}{5!}x^5, -\frac{1}{7!}x^7, \ldots$. Величины n и x вводятся пользователем.

24. Вычислить сумму n первых членов ряда, состоящего из чисел, вычисляемых по формуле $\frac{(2i)!}{4^i (i!)^2 (2i+1)} x^{2i+1}$, i = 0, 1, 2, … . Величины n и x (|x| < 1) вводятся пользователем.
25. Вычислить сумму n первых членов ряда: $x, -\frac{1}{3}x^3, \frac{1}{5}x^5, -\frac{1}{7}x^7, \ldots$. Величины n и x (|x| < 1) вводятся пользователем.
26. Вычислить сумму n первых членов ряда: $x, -\frac{1}{2!}x^2, \frac{1}{4!}x^4, -\frac{1}{6!}x^6, \ldots$. Величины n и x вводятся пользователем.
27. Вычислить произведение тех из введенных n целых чисел, которые больше нуля и без остатка делятся на 3.
28. Вычислить сумму тех из введенных n действительных чисел, которые больше, чем их среднее значение.
29. Дан массив целых чисел размера n. Поменять местами его минимальный и максимальный элементы.
30. Дан массив действительных чисел, содержащий среднемесячную заработную плату сотрудников организации (не более 300). Необходимо вычислить медиану заработной платы, отклонение от медианы для каждого сотрудника, среднее значение заработной платы, отклонение среднего от медианы. Медиана равна значению элемента с индексом n/2, если n – нечетное, или среднему значению элементов с индексами n/2-1, n/2, если n – четное, в массиве, полученном из исходного сортировкой по возрастанию.
31. Даны два массива целых чисел размера n и m соответственно. Переписать в новый массив вторую половину первого массива, а затем - первую треть второго массива.
32. Дан массив действительных чисел размера n. Увеличить все его элементы на величину максимального элемента в массиве.

33. Дан массив целых чисел размера n. Заменить все пары следующих друг за другом одинаковых чисел на одно число, при каждой замене сдвигая «хвост» массива на одну позицию влево и уменьшая счетчик количества элементов массива на 1. Например, из массива 2 2 4 4 1 1 3 следует получить массив 2 4 1 3.
34. Дан массив целых чисел размера n. Исключить из массива повторяющиеся элементы. Не использовать дополнительный массив.
35. Реализовать ввод элементов квадратной матрицы действительных чисел размера n×n.
36. Дана матрица действительных чисел размера n×m. Отсортировать элементы ее столбцов в порядке убывания.
37. Дана матрица действительных чисел размера n×m. Поменять местами два столбца матрицы. Индексы столбцов задаются пользователем.
38. Дана матрица действительных чисел размера n×m. Транспонировать матрицу, записав результат в новую матрицу.
39. Дана матрица действительных чисел размера n×m. Вывести ее на экран в обычном виде (в виде матрицы).
40. Дана квадратная матрица целых чисел размера n×n. Вычислить след матрицы (сумму элементов главной диагонали).

4. Существование объектов, трансляция программы. Создание проекта в среде Microsoft Visual Studio

4.1. Область действия и время жизни объектов

Каждый элемент данных в программе характеризуется областью действия и временем жизни (это относится не только к данным, но и к любому объекту, имеющему идентификатор).

Область действия – участок программы, в котором есть объявление идентификатора и в пределах которого к этому идентификатору можно обратиться. Элемент данных может быть доступным (видимым) на некотором ограниченном участке программы или из любой точки программы. Все зависит от того, в каком месте он объявлен.

Различают четыре вида областей действия: функция, файл, блок (составной оператор) и прототип функции. Прототип функции – это объявление функции, которое задает типы ее параметров, он представляет собой заголовок функции без тела. Напомним, что тело функции – тоже блок.

Если идентификатор объявлен вне любого блока и списка параметров функции (т.е. не в заголовке функции), то область его действия – это весь файл, содержащий программу, начиная с точки, следующей за его объявлением:

```
#include <stdio.h>
int r;
void main() {
  scanf("%d",&r);
}
double u;

int my_func() {
  u = r;
}
```

В данном примере переменная r доступна из функций main и my_func, а переменная u – недоступна (без дополнительного объявления в main) из функции main, так как объявлена после main, но доступна из my_func.

Если идентификатор объявлен внутри блока или внутри списка параметров функции, то область его действия – соответствующий блок. Если же идентификатор объявлен внутри прототипа функции, то область его действия ограничена данным прототипом. Пример:

```
#include <stdio.h>
int main() {
   double my_func(double r1);  //прототип функции my_func
   double r;
   scanf("%lf",&r);     //вводится рост в метрах
   if (r > 1.2) {      //если рост больше, чем 1.2 м
      double t;
```

```
        t = r/0.3048;   //вычисляется рост в футах
        printf("\n%.1lf",t);
    }
    else {
      r = my_func( r ); //иначе вычисляется рост в дюймах
      printf("\n%.1lf",r);
    }
}
double my_func(double r1) {
    return r1/0.3048*12;
}
```

В данном примере переменная r доступна из любой точки функции main, а переменная t – только в блоке: { double t; t = r/0.3048; printf("\n%.1lf",t); }, так как она объявлена в этом блоке. Переменная r1, указанная в прототипе функции my_func, т.е. в первом операторе функции main: double my_func(double r1);, видна только внутри данного прототипа. А переменная r1 из определения функции my_func:

```
double my_func(double r1) {
    return r1/0.3048*12;
}
```

доступна внутри функции my_func, так как указана в списке её параметров.

Если идентификатор, объявленный во внутреннем блоке, совпадает с идентификатором, объявленным во внешнем блоке, охватывающем данный блок, то идентификатор из внешнего блока становится невидимым во внутреннем блоке. При этом говорят, что внутреннее объявление перекрывает внешнее:

```
{
    int r;
    {
        int r, t1;
        …
    }
}
```

В представленном выше примере внутреннее объявление переменной r перекрывает внешнее, и внешняя переменная r невидима в блоке

```
{ int r, t1; … }
```

Иначе говоря, каждый блок имеет свою область видимости. Если в каком-либо вложенном блоке есть переменная, имя которой совпадает с именем лю-

бой переменной из блока, охватывающего данный, то во время выполнения операторов вложенного блока видимой является именно она.

Время жизни объекта – это часть времени работы программы, когда гарантируются нахождение объекта в памяти по определенному адресу и возможность получения последнего сохраненного значения объекта.

Объект с локальным (автоматическим) временем жизни получает новую память при каждом входе в блок, в котором он определен или объявлен. Когда блок завершается, локальный объект исчезает, также исчезает его значение. Для объекта с локальным временем жизни память выделяется, когда блок, в котором был объявлен объект, становится активным (начинают выполняться операторы блока), и освобождается при выходе из него любым способом (переход, возврат, достижение конца). Единственным исключением являются VLA (массивы переменного размера); память под них выделяется при выполнении объявления, а не при входе в блок, и освобождается, когда объявление выходит из области видимости, а не при выходе из блока (начиная с C99). Если вход в блок выполняется многократно, память выделяется многократно.

Объект с глобальным (статическим) временем жизни характеризуется определенной памятью и значением на протяжении всей жизни программы. В случае глобального времени жизни длительность хранения - полное выполнение программы, а значение, хранимое в объекте, инициализируется только один раз, перед основной функцией.

Одинаковые идентификаторы, объявленные в разных или одной областях действия, могут указывать на один и тот же объект (функцию) с помощью связывания. Иначе говоря, несколько одинаковых имен в программе могут относиться к одному единственному объекту. Например, к одной глобальной переменной могут быть обращения в нескольких модулях (файлах) программы. Связывание может быть внешним (в пределах всех транслируемых модулей), внутренним (в пределах одного транслируемого модуля) или может отсутствовать. Виды связывания описаны ниже, после рассмотрения классов хранения.

В языке Си имеется несколько классов хранения, которые определяют время жизни и связывание объектов:

- auto – автоматическое время жизни и отсутствие связывания; применяется к переменным, область действия которых - блок, но не списки параметров функций;
- register – автоматическое время жизни и отсутствие связывания; подсказка оптимизатору сохранить значение переменной в регистре ЦП; применяется к переменным, область действия которых – блок, а также к переменным в списках параметров функций; адрес переменной с таким классом нельзя получить;
- static – статическое время жизни и внутреннее связывание; можно использовать в объявлениях и определениях функций в области действия файла и с переменными в области действия файла и блока, но не в списках параметров функции.
- extern - статическое время жизни и внешнее связывание; может использоваться в объявлениях функций и объектов как в области файлов, так и в области блоков (исключая списки параметров функции). Если extern появляется при повторном объявлении идентификатора, который уже был объявлен с внутренним связыванием, связывание остается внутренним, в противном случае оно внешнее.
- _Thread_local – указывает время жизни потока – части программы, выполняющейся параллельно с другими ее потоками (в данном пособии вопросы, связанные с потоками, не рассматриваются).

В случае отсутствия связывания на идентификатор можно ссылаться только из области, в которой он находится. Все параметры функции и все не являющиеся внешними переменные области блока (включая объявленные как статические) имеют этот тип связывания. В случае внутреннего связывания на идентификатор можно ссылаться из всех областей в текущем файле. Все статические идентификаторы области файла (и функции, и переменные) имеют этот

тип связывания. В случае внешнего связывания на идентификатор можно ссылаться из любых других файлов во всей программе. Все нестатические функции, все внешние переменные (если ранее не были объявлены статическими) и все нестатические переменные области файла имеют этот тип связывания.

Класс хранения может быть указан в объявлении перед именем типа:

```
static int t;
```

Если класс хранения в объявлении не указан, то по умолчанию применяется extern для всех функций, extern для объектов в области видимости файла, auto для объектов в области видимости блока. Для любой структуры или объединения, объявленных с помощью спецификатора класса хранения, время жизни (но не связывание) применяется ко всем их элементам, рекурсивно. Функции не могут быть определены в области действия блока. В объявлениях функций в области действия блока может использоваться extern или вообще не использоваться. В объявлениях функций в области действия файлов могут использоваться extern или static. В параметры функции не могут использовать никакие спецификаторы классов хранения, кроме register. Все параметры функции и нестатические объекты области блока имеют локальное время жизни, а также составные литералы, используемые в блоке.

Все объекты, объявленные как статические, и все объекты с внутренним или внешним связыванием, не объявленные как _Thread_local, имеют глобальное время жизни.

4.2. Этапы трансляции

Трансляция программы, написанной на языке Си, предполагает последовательное выполнение следующих этапов (фаз):

1. При необходимости многобайтные символы физического файла исходной программы (который является текстовым файлом в одной из многобайтных кодировок) отображаются в определенном реализацией порядке в символы

исходного набора, в частности, зависящие от типа операционной системы индикаторы конца строки заменяются на символы новой строки. Триграфы (последовательности из трех символов, каждый из которых присутствует в кодировке ISO 646:1983) заменяются на соответствующие односимвольные представления.

2. Каждый обратный слэш (\) в конце строки, непосредственно предшествующий символу новой строки, удаляется вместе с символом новой строки, и физические строки программы «склеиваются» в последовательность логических строк. Только последний обратный слэш каждой строки может быть частью такого склеивания, что связано с особенностью его использования в конце строки: программист применяет этот символ, чтобы перенести часть длинной строки/идентификатора или другого элемента на следующую строку. Последняя строка программы не должна содержать незавершенные элементы. Если в непустом исходном файле после выполнения операции «склеивания» нет символа новой строки, т.е. если в конце последней строки программного кода был обратный слэш или отсутствовал символ новой строки, результат не определен.
3. Исходный файл разбивается на комментарии, последовательности пробельных символов (к которым относятся пробел, горизонтальная табуляция, вертикальная табуляция, новая строка, новая страница) и специальные неделимые единицы, препроцессируемые токены, к которым относятся:
   - имена заголовочных файлов: <stdlib.h> или "myheader.h";
   - идентификаторы;
   - препроцессируемые числа, в множество которых входят целые и вещественные константы;
   - символьные константы и строковые литералы;
   - знаки операций и пунктуации, такие как *=, + или ##;
   - отдельные непробельные символы, не попадающие в любые другие категории;

Исходный файл не должен заканчиваться частью токена или частью комментария. Каждый комментарий заменяется на один пробел. Символы новой строки сохраняются. Сохраняется ли каждая последовательность пробельных символов, отличная от символа новой строки, или она заменяется на один пробел, определяется реализацией. Если исходный текст уже обработан (разобран на части) до некоторого символа, следующий токен представляет собой максимально возможную последовательность символов, начиная с данного, даже если обработка этого токена приведет к ошибке. Есть лишь одно исключение – названия заголовочных файлов встречаются только в директиве препроцессора #include и в определенных реализацией прагмах #pragma.

4. Препроцессор выполняет все заданные действия. Директива препроцессора #include приводит к рекурсивному выполнению шагов 1-4 трансляции для соответствующего заголовочного или исходного файла. Все директивы препроцессора после выполнения удаляются из исходного файла.
5. Каждый элемент исходного набора символов и escape-последовательности в символьных константах и строковых литералах преобразуются в соответствующие элементы исполняемого набора символов. Если соответствующего элемента нет, элемент преобразуется в определенный реализацией ненулевой [расширенный] символ. Замечание: в некоторых реализациях преобразование может контролироваться опциями командной строки: -finput-кодировка для указания кодировки исходного набора, -fexec-кодировка и fwide-exec-кодировка – для указания кодировки исполняемого набора в символьных константах и строковых литералах, не имеющих префиксов.
6. Соседние строковые литералы сцепляются.
7. Выполняется компиляция. Токены синтаксически и семантически анализируются и транслируются как единицы трансляции.
8. Выполняется компоновка (линкование). Ссылки на все внешние объекты и функции разрешаются. Добавляются необходимые компоненты библиотек

для разрешения оставшихся неопределенными ссылок. Результат трансляции собирается в образ программы, содержащий информацию, необходимую для ее загрузки в соответствующей среде.

4.3. Создание проекта в среде Microsoft Visual Studio

Для создания, сборки и запуска консольной программы на языке Си (C++) следует запустить Microsoft Visual Studio .Net и выполнить следующие шаги:

1. Создать проект, выбрав последовательно в меню «Файл» («File») пункты «Создать» («New»), «Проект» («Project»).
2. В окне типов проектов Visual C++ выбрать Win32, «Консольное приложение Win32» («Win32 Console Application»).
3. Ввести название проекта и нажать «ОК» для его создания.
4. В окне мастера приложений Win32 нажать «Далее» («Next»), выбрать тип «Пустой проект» («Empty Project»), затем нажать «Готово» («Done»).
5. Открыть окно «Обозреватель решений» («Solution explorer») из меню «Вид» («View»), если оно не было открыто.
6. Добавить новый файл для написания программного кода следующим образом:

   а) В окне «Обозреватель решений» правой кнопкой мыши щелкнуть папку «Исходные файлы», выбрать «Добавить» («Add»), затем – «Новый элемент» («New Item»).

   б) В узле «Код» («Code») появившегося дерева узлов выбрать «Файл C++ (.cpp)», ввести имя файла, нажать «Добавить» («Add»).

   Файл исходного кода появится в папке «Исходные файлы» («Source Files») в окне «Обозреватель решений» и откроется в редакторе Visual Studio.
7. В созданном файле исходного кода разместить текст программы, сохранить файл.

8. В меню «Построение» («Build») выбрать «Построить решение» («Build solution»). Информация о компиляции выводится в окне «Выходные данные».
9. Выбрать «Запуск без отладки» в меню «Отладка» («Debugging»).

4.4. Контрольные вопросы и упражнения

Контрольные вопросы

1. Что такое область действия (видимости) объекта?
2. Перечислите возможные области действия.
3. Что такое время жизни объекта?
4. Чем отличается автоматическое время жизни объекта от статического?
5. Перечислите классы хранения объектов и объясните, на что они указывают.
6. Какие виды связывания существуют?
7. Можно ли определить статическую переменную?
8. Можно ли определить функцию в блоке?
9. Можно ли определить переменную с одним и тем же именем во внешнем и внутреннем по отношению к упомянутому внешнему блоках?
10. Чем отличается определение функции с классом хранения static от объявления функции с классом хранения extern.
11. Какой класс хранения у переменной, впервые объявленной в блоке?
12. Какие действия выполняются на первых двух этапах трансляции?
13. Что делает препроцессор?
14. Какие элементы удаляются из программы после того, как препроцессор закончит работу?
15. Что происходит на этапах компиляции и компоновки?
16. Назовите основные этапы создания консольных программ в среде Microsoft Visual Studio.

Упражнения

1. Создайте объявление функции, область действия которого распространяется только на файл, в котором оно создано.
2. Определите функцию, в которой есть объявления глобальных переменных и обращения к ним.
3. Создайте цикл для подсчета суммы введенных целых чисел со счетчиком, имеющим локальное для данного цикла время жизни.
4. Создайте переменную с глобальным временем жизни и покажите, как на нее сослаться из других транслируемых единиц.
5. Дан фрагмент программы:

```
{
   int r = 3;
   {
      int r = 1, t1 = 7;
      for (int i = 1; i <= t1; ++i) printf("\n%d",r++);
   }
   printf("\n%d",r);
}
```

   Что будет выведено в консоль в результате его работы?
6. Создайте пустой и непустой консольные проекты в среде Microsoft Visual Studio. Поясните, чем они отличаются друг от друга.

Библиографический список

1. Пратт, Т. Языки программирования: разработка и реализация : под ред. А. Матросова / Т. Пратт, М.Зелковиц. – Санкт-Петербург: Питер, 2002. – 688 с. – Текст: непосредственный
2. Давыдов, В.Г. Программирование и основы алгоритмизации : учеб. пособие / В.Г. Давыдов. – Москва: Высшая школа, 2003. - 447 с. – Текст: непосредственный

3. Паттерсон, Д. Архитектура компьютера и проектирование компьютерных систем : Классика Computers Science / Д. Паттерсон, Дж. Хеннеси. – 4-е изд. – Санкт-Петербург: Питер, 2012. – 784 с. – Текст: непосредственный
4. Уэйт, М. Язык Си : руководство для начинающих / пер. с англ. Л. Н. Горинович и В. С. Явниловича под ред. д-ра техн. наук Э. А. Трахтенгерца / М. Уэйт, С. Прата, Д. Мартин. – Москва: Мир, 1988. – 512 с. – Текст: непосредственный
5. C language. – Режим доступа: https://en.cppreference.com/w/c/language, свободный. – Яз. англ. (дата обращения 07.06.2019). – Текст: электронный

Учебное издание

Журавлева Марина Гарриевна,
Алексеев Владимир Александрович,
Домашнев Павел Алексеевич

ОСНОВЫ ПРОГРАММИРОВАНИЯ. ВВЕДЕНИЕ В ЯЗЫК СИ. ЧАСТЬ 1

УЧЕБНОЕ ПОСОБИЕ
по курсам «Программирование»,
«Основы алгоритмизации и программирования»

Редактор Е.А. Федюшина
Подписано в печать Формат 60х84 1/16. Бумага офсетная. Ризография. Объем 6,2 п.л. Тираж 100 экз. Заказ № _____ .
Издательство Липецкого государственного технического университета.
Полиграфическое подразделение Издательства ЛГТУ.
398055, Липецк, ул. Московская, 30.